EEET 4071 Advanced Control (2021)

Lecture 3: State-space equations

Dr. Yee Wei Law ⟨yeewei.law@unisa.edu.au⟩

Contents

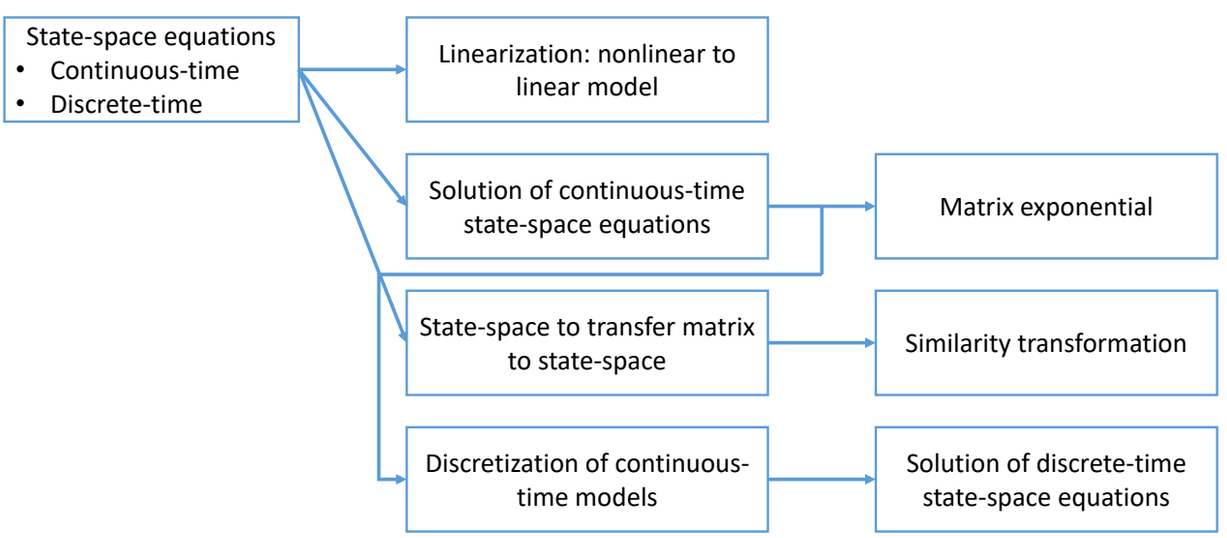

Figure 1: Roadmap for this lecture.

1 Introduction

In classical control, we have three options when synthesizing a controller for a single-input single-output (SISO) plant.

- If the plant does not have a significant time delay, or oscillatory modes (poles on the imaginary axis), or large parameter variations [Cor04], then we can *tune* a PID controller for the plant even if we do not have a model of the plant.
- If we have frequency response data but not a model of the plant, then we can apply the *frequency response* technique.
- If we have a model of the plant, then we can apply the *root locus* technique.

For multivariable plants, we need modern control techniques, and a model of the plant in the form of differential equations, to apply these techniques. It is only with this model, that a stability proof can be derived. The importance of modeling can thus not be overstated. Figure 1 shows the roadmap for this lecture.

2 State-space equations

A plant is a *dynamical system*, i.e., a system whose behavior evolves with time, either by itself (autonomously) or under the influence of external processes. A dynamical system is described by one or more differential equations. The idea of state-space modeling is to "package" these differential equations into two first-order differential equations in the vector-matrix form to facilitate analysis:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \tag{1}$$
$$\boldsymbol{y}(t) = \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t)). \tag{2}$$

Eq. (1) is called a *state equation* or *state differential equation*, whereas $\boldsymbol{x}(t)$ is a vector of states, called a *state vector*. The vector $\boldsymbol{u}(t)$ represents the inputs to the system, and if $\boldsymbol{u}(t) = \boldsymbol{0}$, the system is called an *autonomous system*. Eq. (2) is called an *output equation* or *observation equation* or *measurement equation*, which relates the output vector $\boldsymbol{y}(t)$ to the state and input vectors. Whereas the states and the inputs are largely determined by the nature of the system, there is flexibility in the choice of the outputs. Together, Equations (1) and (2) are called *state-space equations*.

Example 1

Suppose a system is described by the second-order differential equations

$$\ddot{x}_1 + x_2^2 = 1,$$
$$\ddot{x}_2 + \dot{x}_1 = t,$$

where x_1 and x_2 are time-dependent variables. Rearranging the terms so that only the highest-order terms stay on the left-hand side, we have

$$\ddot{x}_1 = -x_2^2 + 1,$$
$$\ddot{x}_2 = -\dot{x}_1 + t.$$

Letting $\boldsymbol{z} = [\dot{x}_1 \quad \dot{x}_2 \quad x_2]^\top$, and $u = t$, we can then write the state equation as

$$\dot{\boldsymbol{z}} = \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\left(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \boldsymbol{z}\right)^2 + 0 \cdot u + 1 \\ \begin{bmatrix} -1 & 0 & 0 \end{bmatrix} \boldsymbol{z} + u \\ \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \boldsymbol{z} + 0 \cdot u \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{z}, u) \\ f_2(\boldsymbol{z}, u) \\ f_3(\boldsymbol{z}, u) \end{bmatrix}.$$

Note that f_1 is nonlinear, so the state equation is nonlinear. Also note that if we swap the order of $\dot{x}_1$, $\dot{x}_2$, x_2 in $\boldsymbol{z}$, and modify f_1, f_2, f_3 accordingly, then we will get an equivalent *state-space representation* of the same system. Thus, state-space representations are not unique. We will revisit this point in Section 5.

We have seen a fictitious model in Example 1. Let us look at an actual model in the next example.

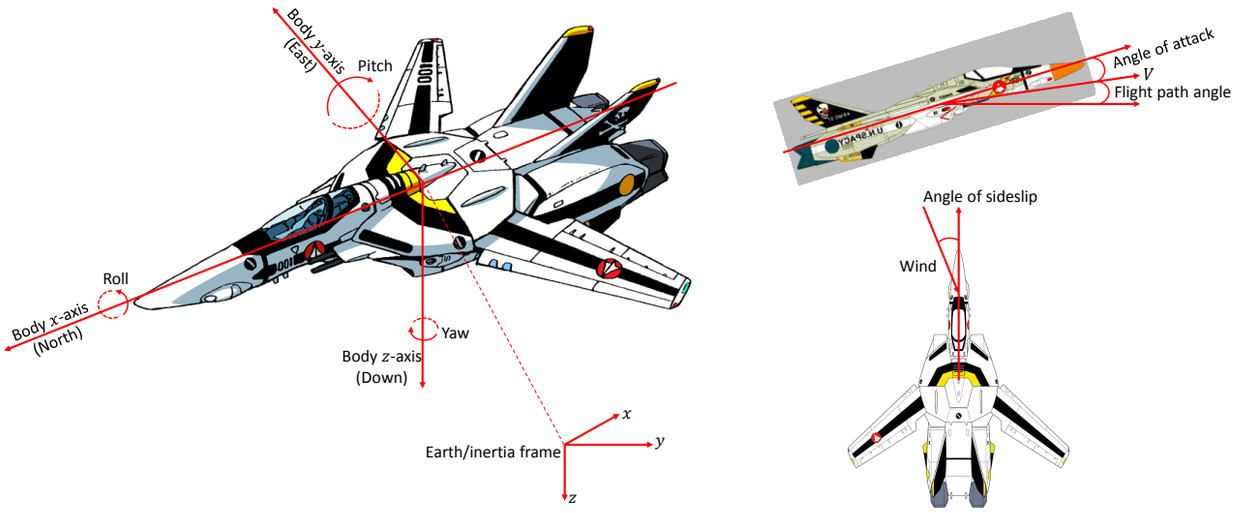

Figure 2: Motion variables of an aircraft with respect to the body frame and earth/inertial frame of an aircraft. The direction of roll/pitch/yaw follows the right-hand rule.

Example 2

The dynamic model, in the form of *equations of motion*, of a fixed-wing aircraft (see Figure 2) is well documented. The state equation derived by NASA [DAP88] takes the form

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{x}}_1 \\ \dot{\boldsymbol{x}}_2 \\ \dot{\boldsymbol{x}}_3 \\ \dot{\boldsymbol{x}}_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_1(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{f}_2(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{f}_3(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{f}_4(\boldsymbol{x}, \boldsymbol{u}) \end{bmatrix},$$

where

- $\boldsymbol{x}_1 = [p \quad q \quad r]^\top$ consists of the roll rate, pitch rate, yaw rate of the aircraft;
- $\boldsymbol{x}_2 = [V \quad \alpha \quad \beta]^\top$ consists of the velocity, angle of attack, angle of sideslip of the aircraft;
- $\boldsymbol{x}_3 = [\psi \quad \theta \quad \phi]^\top$ consists of the Euler angles (yaw, pitch, roll) of the aircraft, representing the successive rotations required to align the earth axes with the body axes [DAP88, p. 8];
- $\boldsymbol{x}_4 = [h \quad x \quad y]^\top$ consists of the altitude, earth x-coordinate, earth y-coordinate of the aircraft.

The 12-state state equation is quite involved, and for illustrative purposes, let us just look at $\boldsymbol{f}_1$:

$$\boldsymbol{f}_1(\boldsymbol{x}, \boldsymbol{u}) = -\mathbf{J}^{-1}\left([\mathbf{I}_3 \quad \mathbf{0} \quad \mathbf{0} \quad \mathbf{0}]\boldsymbol{x} \times \mathbf{J}[\mathbf{I}_3 \quad \mathbf{0} \quad \mathbf{0} \quad \mathbf{0}]\boldsymbol{x}\right) + \mathbf{J}^{-1}\begin{bmatrix} L \\ M \\ N \end{bmatrix},$$

which can be derived from Euler's equation of motion, where

- $\mathbf{J}$ is the inertia matrix of the aircraft;
- $\mathbf{I}_3$ is the 3-by-3 identity matrix;
- $\times$ is the cross product operator;
- L, M, N are the total torques about the body x, y, z axes due to aerodynamics and propulsion.

The vector functions $\boldsymbol{f}_2$, $\boldsymbol{f}_3$, $\boldsymbol{f}_4$ can be derived following [DAP88].

The nonlinear systems in Examples 1 and 2 are more difficult than linear systems to analyze and design controllers for, so the standard practice is to *linearize* a nonlinear system, before considering nonlinear techniques. Before we look at linearization techniques, let us study linear state-space equations. If the system differential equations are all linear, then we can "package" the differential equations into linear state-space equations:

$$\dot{\boldsymbol{x}}(t) = \mathbf{A}(t)\boldsymbol{x}(t) + \mathbf{B}(t)\boldsymbol{u}(t), \tag{3}$$
$$\boldsymbol{y}(t) = \mathbf{C}(t)\boldsymbol{x}(t) + \mathbf{D}(t)\boldsymbol{u}(t), \tag{4}$$

If n is the number of states, m is the number of inputs, p is the number of outputs, then

- $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the *system / dynamic / evolution matrix*;
- $\mathbf{B} \in \mathbb{R}^{n\times m}$ is the *input / control matrix*;
- $\mathbf{C} \in \mathbb{R}^{p\times n}$ is the *output / observation matrix*;
- $\mathbf{D} \in \mathbb{R}^{p\times m}$ is the *coupling / direct feedthrough / direct transmission / direct transfer / feedforward matrix*.

If furthermore the linear system is time-invariant, i.e., the matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are time-independent, then we can write

$$\dot{\boldsymbol{x}}(t) = \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t), \tag{5}$$
$$\boldsymbol{y}(t) = \mathbf{C}\boldsymbol{x}(t) + \mathbf{D}\boldsymbol{u}(t). \tag{6}$$

In discrete time, the state-space equations for linear time-invariant (LTI) systems take the form

$$\boldsymbol{x}[k+1] = \mathbf{F}\boldsymbol{x}[k] + \mathbf{G}\boldsymbol{u}[k], \tag{7}$$
$$\boldsymbol{y}[k] = \mathbf{C}\boldsymbol{x}[k] + \mathbf{D}\boldsymbol{u}[k]. \tag{8}$$

Note that Eq. (7) expresses a recursive relationship between the state at the next time instant and the state at the current time instant. In Sect. 6, we will learn how to convert a model from continuous-time to discrete-time.

3 Model linearization

Most physical systems are nonlinear, but we can *linearize* their models to facilitate analysis and controller design. The scientific basis for the validity of this approach is provided by Lyapunov's first/indirect method. For now, let us focus on the procedure. Linearization refers to the linear approximation of a model in the neighborhood of a chosen operating point using Taylor's series expansion. Consider again the continuous-time state-space equations (1)–(2), and let $\boldsymbol{u}_e(t)$ be an input that leads the system to equilibrium state/point $\boldsymbol{x}_e$, i.e.,

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}_e, \boldsymbol{u}_e(t), t) = \mathbf{0},$$

where $\boldsymbol{f}$ is a nonlinear but linearizable function.

The equilibrium state is by definition the state in which the system has zero $\dot{\boldsymbol{x}}$. If not explicitly specified, the system input is assumed to be zero when the system is in its equilibrium state.

Suppose the input is perturbed slightly from $\boldsymbol{u}_e(t)$ to $\boldsymbol{u}_e(t) + \delta_{\boldsymbol{u}}(t)$. Consequently, the state is perturbed slightly from $\boldsymbol{x}_e$ to $\boldsymbol{x}_e + \delta_{\boldsymbol{x}}(t)$. Then,

$$\dot{\boldsymbol{x}}(t) = \dot{\boldsymbol{x}}_e + \dot{\delta}_{\boldsymbol{x}}(t) = \dot{\delta}_{\boldsymbol{x}}(t).$$

Using Taylor's series expansion, we can write (dropping the (t) parts for clarity)

$$\begin{aligned} \dot{\boldsymbol{x}} = \dot{\delta}_{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x}_e + \delta_{\boldsymbol{x}}, \boldsymbol{u}_e + \delta_{\boldsymbol{u}}, t) \\ &= \boldsymbol{f}(\boldsymbol{x}_e, \boldsymbol{u}_e, t) + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_e \\ \boldsymbol{u}=\boldsymbol{u}_e}} \delta_{\boldsymbol{x}} + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_e \\ \boldsymbol{u}=\boldsymbol{u}_e}} \delta_{\boldsymbol{u}} + \text{higher-order terms}, \end{aligned} \tag{9}$$

where $\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is a shorthand for the *Jacobian*:

$$\frac{\partial(f_1, \ldots, f_n)}{\partial(x_1, \ldots, x_n)} \overset{\text{def}}{=} \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix}.$$

Note $\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is a square matrix.

Quiz 1

Based on how $\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is defined, can you tell how $\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}$ is defined? Furthermore, what is the dimension of $\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}$?

Treating $\delta_{\boldsymbol{x}}(t)$ as the state vector, and ignoring the higher-order terms in Eq. (9), we have the following linearized state equation:

$$\dot{\delta}_{\boldsymbol{x}}(t) = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_e \\ \boldsymbol{u}=\boldsymbol{u}_e}} \delta_{\boldsymbol{x}}(t) + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_e \\ \boldsymbol{u}=\boldsymbol{u}_e}} \delta_{\boldsymbol{u}}(t), \tag{10}$$

where $\delta_{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - \boldsymbol{x}_e$ and $\delta_{\boldsymbol{u}}(t) = \boldsymbol{u}(t) - \boldsymbol{u}_e$. $\delta_{\boldsymbol{x}}$ is often written as $\Delta \boldsymbol{x}$.

Example 3

James Maxwell (the physicist who gave us Maxwell's equations for electromagnetism) co-founded the theory of automatic control in his 1868 paper "On governors," in which he analyzed the stability of flyball governors (see Figure 3 and [Żak03, Section 1.7.1, Section 2.4, Example 4.6]). A governor is a mechanical device that uses mainly the properties of centrifugal force to measure and regulate the speed of a machine. An enlightening demonstration of how a diesel engine governor works can be found on YouTube: https://youtu.be/OiHb2L8ei8E.

Let $\boldsymbol{x} = [\varphi \quad \dot{\varphi} \quad \omega]^\top$, where φ is the angle between an arm with the vertical line, and ω is the angular speed of the flywheel. ω is related to the angular speed of the flyball by

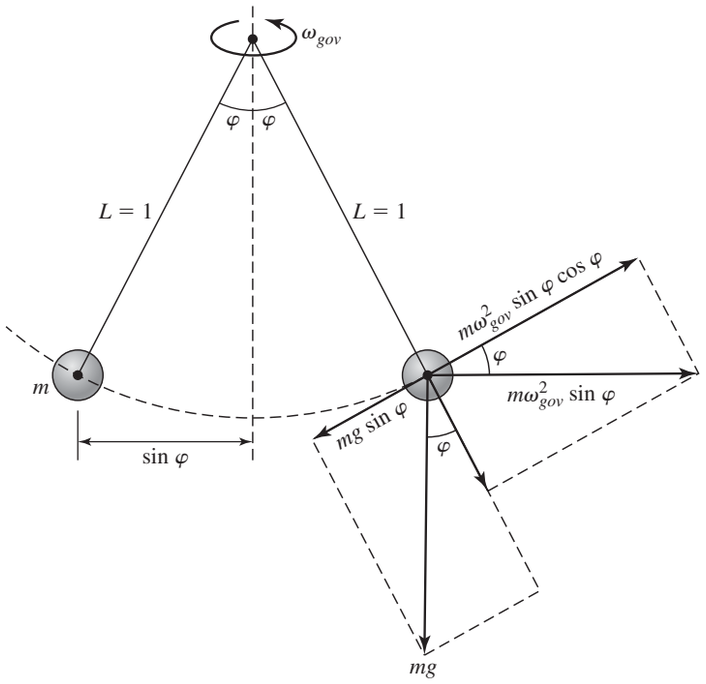

Figure 3: A flyball governor and the forces acting on it.

$\omega = \omega_{gov}/N$, where N is the gear ratio. The nonlinear state equation is

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\varphi} \\ \ddot{\varphi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ f_3(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \dot{\varphi} \\ \dfrac{1}{2}N^2\omega^2 \sin(2\varphi) - g\sin(\varphi) - \dfrac{b}{m}\dot{\varphi} \\ \dfrac{\kappa}{J}\cos(\varphi) - \dfrac{\tau}{J} \end{bmatrix}, \tag{11}$$

where b is the coefficient of friction; J is the moment of inertia of the flywheel; κ and τ are some other constants. For simplicity, we are not dealing with any input.

The linearization process starts by determining the equilibrium state of Eq. (11):

$$\dot{\boldsymbol{x}} = 0 \implies \begin{cases} \dot{\varphi}_e = 0, \\ \ddot{\varphi}_e = \dfrac{1}{2}N^2\omega_e^2 \sin(2\varphi_e) - g\sin(\varphi_e) - \dfrac{b}{m}\dot{\varphi}_e = \left(N^2\omega_e^2\cos(\varphi_e) - g\right)\sin(\varphi_e) = 0, \\ \dot{\omega}_e = \dfrac{\kappa}{J}\cos(\varphi_e) - \dfrac{\tau}{J} = \dfrac{1}{J}(\kappa\cos(\varphi_e) - \tau) = 0, \end{cases}$$

$$\implies \begin{cases} \dot{\varphi}_e = 0, \\ \omega_e = \sqrt{\dfrac{g}{N^2\cos(\varphi_e)}} = \sqrt{\dfrac{g\kappa}{N^2\tau}}, \\ \varphi_e = \arccos\left(\dfrac{\tau}{\kappa}\right). \end{cases}$$

Above, we disregarded the possibility that $\sin(\varphi_e) = 0$ since that contradicts $\cos(\varphi_e) = \tau/\kappa$. Therefore the equilibrium states are $\varphi_e = \arccos\left(\dfrac{\tau}{\kappa}\right)$, $\dot{\varphi}_e = 0$, $\omega_e = \sqrt{\dfrac{g\kappa}{N^2\tau}}$.

Next, we calculate the following partial derivatives for the Jacobian:

$$\frac{\partial f_1}{\partial \varphi} = \frac{\partial \dot{\varphi}}{\partial \varphi} = 0,$$
$$\frac{\partial f_1}{\partial \dot{\varphi}} = \frac{\partial \dot{\varphi}}{\partial \dot{\varphi}} = 1,$$
$$\frac{\partial f_1}{\partial \omega} = \frac{\partial \dot{\varphi}}{\partial \omega} = 0,$$

$$\frac{\partial f_2}{\partial \varphi} = \frac{\partial}{\partial \varphi}\left\{\frac{1}{2}N^2\omega^2\sin(2\varphi) - g\sin(\varphi) - \frac{b}{m}\dot{\varphi}\right\} = N^2\omega^2\cos(2\varphi) - g\cos(\varphi),$$
$$\frac{\partial f_2}{\partial \dot{\varphi}} = \frac{\partial}{\partial \dot{\varphi}}\left\{\frac{1}{2}N^2\omega^2\sin(2\varphi) - g\sin(\varphi) - \frac{b}{m}\dot{\varphi}\right\} = -\frac{b}{m},$$
$$\frac{\partial f_2}{\partial \omega} = \frac{\partial}{\partial \omega}\left\{\frac{1}{2}N^2\omega^2\sin(2\varphi) - g\sin(\varphi) - \frac{b}{m}\dot{\varphi}\right\} = N^2\omega\sin(2\varphi),$$

$$\frac{\partial f_3}{\partial \varphi} = \frac{\partial}{\partial \varphi}\left\{\frac{\kappa}{J}\cos(\varphi) - \frac{\tau}{J}\right\} = -\frac{\kappa}{J}\sin(\varphi),$$
$$\frac{\partial f_3}{\partial \dot{\varphi}} = \frac{\partial}{\partial \dot{\varphi}}\left\{\frac{\kappa}{J}\cos(\varphi) - \frac{\tau}{J}\right\} = 0,$$
$$\frac{\partial f_3}{\partial \omega} = \frac{\partial}{\partial \omega}\left\{\frac{\kappa}{J}\cos(\varphi) - \frac{\tau}{J}\right\} = 0.$$

Now we assemble all the partial derivatives into the Jacobian:

$$\left.\frac{\partial(f_1, f_2, f_3)}{\partial(\varphi, \dot{\varphi}, \omega)}\right|_{\varphi_e, \dot{\varphi}_e, \omega_e} = \begin{bmatrix} \frac{\partial f_1}{\partial \varphi} & \frac{\partial f_1}{\partial \dot{\varphi}} & \frac{\partial f_1}{\partial \omega} \\ \frac{\partial f_2}{\partial \varphi} & \frac{\partial f_2}{\partial \dot{\varphi}} & \frac{\partial f_2}{\partial \omega} \\ \frac{\partial f_3}{\partial \varphi} & \frac{\partial f_3}{\partial \dot{\varphi}} & \frac{\partial f_3}{\partial \omega} \end{bmatrix}_{\varphi_e, \dot{\varphi}_e, \omega_e}$$
$$= \begin{bmatrix} 0 & 1 & 0 \\ N^2\omega_e^2\cos(2\varphi_e) - g\cos(\varphi_e) & -\frac{b}{m} & N^2\omega_e\sin(2\varphi_e) \\ -\frac{\kappa}{J}\sin(\varphi_e) & 0 & 0 \end{bmatrix}.$$

Let us simplify a few elements in the Jacobian:

$$N^2\omega_e^2\cos(2\varphi_e) - g\cos(\varphi_e) = N^2\omega_e^2(2\cos^2(\varphi_e) - 1) - g\cos(\varphi_e)$$
$$= N^2\left(\frac{g\kappa}{N^2\tau}\right)\left(\frac{2\tau^2 - \kappa^2}{\kappa^2}\right) - g\frac{\tau}{\kappa}$$
$$= \frac{g}{\tau\kappa}(\tau^2 - \kappa^2).$$

$$N^2\omega_e\sin(2\varphi_e) = N^2\sqrt{\frac{g\kappa}{N^2\tau}}(2)\sin(\varphi_e)\cos(\varphi_e) = 2N\sqrt{\frac{g\kappa}{\tau}}\sqrt{\frac{\kappa^2 - \tau^2}{\kappa^2}}\sqrt{\frac{\tau^2}{\kappa^2}}$$
$$= 2N\sqrt{\frac{g\tau(\kappa^2 - \tau^2)}{\kappa^3}}.$$
$$-\frac{\kappa}{J}\sin(\varphi_e) = -\frac{\kappa}{J}\sqrt{\frac{\kappa^2 - \tau^2}{\kappa^2}} = -\frac{\sqrt{\kappa^2 - \tau^2}}{J}.$$

Finally, the linearized state equation is

$$\Delta\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{g}{\tau\kappa}(\tau^2 - \kappa^2) & -\frac{b}{m} & 2N\sqrt{\frac{g\tau(\kappa^2 - \tau^2)}{\kappa^3}} \\ -\frac{\sqrt{\kappa^2 - \tau^2}}{J} & 0 & 0 \end{bmatrix}\Delta\boldsymbol{x}.$$

The linearized state equation can then be used to analyze the stability of the governor so that proper values of b, J, κ, τ or some other parameters can be determined, but stability analysis is discussed in a later lecture.

After seeing how we can perform linearization manually in the previous example, let us see how we can perform linearization programmatically — specifically with MATLAB, as this is how we tend to perform linearization in practice. MATLAB comes with a number of examples on linearization: https://www.mathworks.com/discovery/linearization.html. While the examples are useful, a more systematic approach would be to look at the tools available in the many toolboxes, as shown below.

- `jacobian` (command) from Symbolic Math Toolbox
  - **Utility**: For deriving the Jacobians of a symbolic state equation.
  - **Significant links**:
    * https://www.mathworks.com/help/symbolic/jacobian.html
- Control System Tuner (app) from Control System Toolbox
  - **Utility**: When used with a Simulink model, the Control System Tuner app is accessible via the "Analysis" menu. The app computes system responses, and tunes the controller parameters for a linearization of the model. By default, Control System Tuner linearizes the model at the *model initial conditions*, but one or more alternate operating points can be specified for tuning the model. Control System Tuner accepts two types of alternate operating points:
    1. *Simulation snapshots*: Control System Tuner simulates the model for a specified amount of time, and linearizes using the state values at that time. Simulation snapshot linearization is useful, for instance, when the model is known to reach an equilibrium state after a certain simulation time.
    2. *Steady-state operating point*: Control System Tuner finds a steady-state operating point at which some specified condition is met (trimming). For example, if the model represents an automobile motor, an operating point can be a point at which the motor operates steadily at 2000 rpm.

    Note the `controlSystemTuner` *command* presents a slightly different GUI that does not support linearization.
  - **Significant links**:
    * https://www.mathworks.com/help/control/ug/specify-operating-point-for-tuning-in-c
      html
    * https://www.mathworks.com/help/control/examples/tune-a-simulink-model-of-a-contro
      html
- `linmod` and `dlinmod` (commands) from Simulink
  - **Utility**: For extracting a continuous-time linear state-space model from a Simulink model around an operating point. `linmod` works by linearizing each block in a model individually. The default algorithm uses preprogrammed analytic block Jacobians for most blocks, resulting theoretically in more accurate linearization than numerical perturbation of block inputs and states. `linmod` handles Transport Delay blocks by replacing the linearization of the blocks with a Padé approximation. `dlinmod` is the discrete-time equivalent of `linmod`.
  - **Significant links**:
    * https://www.mathworks.com/help/simulink/ug/linearizing-models.html
    * https://www.mathworks.com/help/simulink/slref/linmod.html
    * https://www.mathworks.com/help/simulink/slref/dlinmod.html
    * https://www.mathworks.com/help/slcontrol/ug/exact-linearization-algorithm.html
- `linmod2` (command) from Simulink
  - **Utility**: Like `linmod`, for extracting a continuous-time linear state-space model from a Simulink model around an operating point. `linmod2` computes a linear state-space model by perturbing the model inputs and model states, and uses an advanced algorithm to reduce

truncation error. Like `linmod`, `linmod2` also handles Transport Delay blocks by replacing the linearization of the blocks with a Padé approximation.
  - **Significant links**:
    * https://www.mathworks.com/help/simulink/ug/linearizing-models.html
    * https://www.mathworks.com/help/simulink/slref/linmod2.html

- `linmodv5` (command) from Simulink
  - **Utility**: Like `linmod`, for extracting a continuous-time linear state-space model from a Simulink model around an operating point. However, `linmodv5` is a legacy command created prior to MATLAB version 5.3, and should be avoided.
  - **Significant links**:
    * https://www.mathworks.com/help/simulink/ug/linearizing-models.html
    * https://www.mathworks.com/help/simulink/slref/linmodv5.html

- `linearize` (command) from Simulink Control Design
  - **Utility**: For linearizing a Simulink model for command-line analysis of poles and zeros, plot responses, and control design. To be used with commands `findop` and `linio`. Supports batch linearization.
  - **Significant links**:
    * https://www.mathworks.com/help/slcontrol/ug/choosing-linearization-tools.html
    * https://www.mathworks.com/help/slcontrol/ug/linearize.html
    * https://www.mathworks.com/help/slcontrol/batch-linearization.html
    * https://www.mathworks.com/help/slcontrol/ug/linearize-at-triggered-simulation-eve
      html
    * https://www.mathworks.com/help/slcontrol/examples/computing-operating-point-snaps
      html

Detail: Batch linearization

Batch linearization refers to extracting multiple linearizations from a model for various combinations of I/Os, operating points, and parameter values. Batch linearization enables the analysis of the time-domain, frequency-domain, and stability characteristics of a Simulink model, or portions of a model, under varying operating conditions and parameter ranges. The results of batch linearization can be used to design controllers that are robust against parameter variations, or to design gain-scheduled controllers for different operating conditions. Batch linearization results can also be used to implement linear parameter-varying (LPV) approximations of nonlinear systems using the "LPV System" block of Control System Toolbox.

- `slLinearize` (command) from Simulink Control Design
  - **Utility**: Provides interface for batch linearization of Simulink models.
  - **Significant links**:
    * https://www.mathworks.com/help/slcontrol/ug/choosing-linearization-tools.html
    * https://www.mathworks.com/help/slcontrol/ug/sllinearizer.html
    * https://www.mathworks.com/help/slcontrol/batch-linearization.html

- Linear Analysis Tool (app) from Simulink Control Design
  - **Utility**: Accessible through the "Analysis" menu of Simulink, this tool enables interactive exploration of Simulink model linearization under different operating conditions. The tool supports diagnosis of linearization problems, and batch linearization for varying model parameter values. It can also generate MATLAB code for batch linearization.
  - **Significant links**:
    * https://www.mathworks.com/help/slcontrol/ug/choosing-linearization-tools.html

    * https://www.mathworks.com/help/slcontrol/ug/linearanalysistool-app.html
    * https://www.mathworks.com/help/slcontrol/ug/linearize-simulink-model.html
    * https://www.mathworks.com/help/slcontrol/ug/linearize-at-simulation-snapshot.html

- Linear Analysis Plots (block) from Simulink Control Design
  - **Utility**: For visualizing the linear characteristics of a Simulink model during simulation.
  - **Significant links**:
    * https://www.mathworks.com/help/slcontrol/ug/choosing-linearization-tools.html
    * https://www.mathworks.com/help/slcontrol/ug/visualize-bode-response-of-simulink-m
      html

For linearizsation, the tools from Simulink Control Design are generally more featureful than the tools from Simulink. Among the tools above, we shall practice using `jacobian` and `linmod` in Practical 3.

Example 4

See accompanying Live Script.

4 Solving state-space equations

Given the state-space equations describing a system, and the initial condition of that system, we shall be able to *solve* the state-space equations to determine how the system evolves with time. For the LTI state-space equations with initial condition $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$,

$$\begin{aligned}\dot{\boldsymbol{x}}(t) &= \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t),\\ \boldsymbol{y}(t) &= \mathbf{C}\boldsymbol{x}(t) + \mathbf{D}\boldsymbol{u}(t);\end{aligned}$$

the standard solution is

$$\boldsymbol{x}(t) = e^{\mathbf{A}(t-t_0)}\boldsymbol{x}_0 + \int_{t_0}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau, \tag{12}$$

$$\boldsymbol{y}(t) = \underbrace{\mathbf{C}e^{\mathbf{A}(t-t_0)}\boldsymbol{x}_0}_{\text{free response}} + \underbrace{\mathbf{C}\int_{t_0}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau + \mathbf{D}\boldsymbol{u}(t)}_{\text{forced response}}. \tag{13}$$

Since we are dealing with LTI systems, we can shift t_0 to zero, without loss of generality. Eqs. (12) and (13) allow us to determine the time response of a continuous-time LTI system, given its initial state and a specification of the input. In the equations, $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$ consist of two terms each:

- a *free response* / *zero-input response* term that depends on the initial state $\boldsymbol{x}_0$, and
- a *forced response* / *zero-state response* term that depends on input $\boldsymbol{u}(t)$ over the period $[t_0, t]$.

Quiz 2

What is yet another synonym for “free response”?

Considering how $e^{\mathbf{A}t}$ links the initial state to the current state, the matrix $e^{\mathbf{A}t}$ is called the *state transition matrix*. The term $e^{\mathbf{A}t}$ is not a regular exponential, but a *matrix exponential*:

$$e^{\mathbf{A}t} \stackrel{\text{def}}{=} \sum_{k=0}^{\infty} \frac{\mathbf{A}^k t^k}{k!}. \tag{14}$$

The matrix exponential has these properties:

- $\frac{d}{dt} e^{\mathbf{A}t} = \mathbf{A} e^{\mathbf{A}t} = e^{\mathbf{A}t} \mathbf{A}$, which is directly verifiable by definition.
- $e^{\mathbf{A}t_1} e^{\mathbf{A}t_2} = e^{\mathbf{A}(t_1+t_2)}$, which is also directly verifiable by definition.
- $(e^{\mathbf{A}t})^{-1} = e^{-\mathbf{A}t}$, which follows from above, and means $e^{\mathbf{A}t}$ is nonsingular.
- However, $e^{\mathbf{A}t} e^{\mathbf{B}t} = e^{(\mathbf{A}+\mathbf{B})t} = e^{\mathbf{B}t} e^{\mathbf{A}t}$ *iff* $\mathbf{AB} = \mathbf{BA}$. Note matrix product is generally not commutative.

Quiz 3

This is Analytic Exercise AE2.2 from [WL07]: Show that for any $n \times n$ matrix $\mathbf{A}$ and any scalar γ,

$$e^{(\gamma \mathbf{I}+\mathbf{A})t} = e^{\gamma t} e^{\mathbf{A}t}.$$

Note that $\mathbf{I}$ is the $n \times n$ identity matrix, and t is the time variable. Hint: Use one of the properties of the matrix exponential.

Quiz 4

In Eq. (12), the matrix $e^{t\mathbf{A}}\mathbf{B}$ is called an *input-to-state impulse matrix*. Can you explain why?

There are many ways to calculate a matrix exponential [ML03], but they are generally tedious to perform by hand.

- MATLAB provides function `expm` for calculating matrix exponentials.
- Later, we will learn how to calculate a matrix exponential by applying the Cayley-Hamilton theorem, a very important theorem in linear algebra. For now, let us first consider some special cases that are more convenient for manual calculation.

Example 5

Suppose $\mathbf{A} = \begin{bmatrix} 0 & \alpha \\ 0 & 0 \end{bmatrix}$. We can easily check that $\mathbf{A}^2 = \mathbf{A}^3 = \cdots = 0$. Such a matrix is so-called *nilpotent*. Then from Eq. (14),

$$e^{\mathbf{A}t} = \mathbf{I} + \mathbf{A}t = \begin{bmatrix} 1 & \alpha t \\ 0 & 1 \end{bmatrix}.$$

Example 6

Suppose $\mathbf{A} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$. We can easily check that $\mathbf{A}^k = \begin{bmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{bmatrix}$. Then from Eq. (14),

$$e^{\mathbf{A}t} = \sum_{k=0}^{\infty} \frac{\mathbf{A}^k t^k}{k!} = \sum_{k=0}^{\infty} \frac{t^k}{k!} \begin{bmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{bmatrix} = \sum_{k=0}^{\infty} \begin{bmatrix} \frac{\lambda_1^k t^k}{k!} & 0 \\ 0 & \frac{\lambda_2^k t^k}{k!} \end{bmatrix} = \begin{bmatrix} e^{\lambda_1 t} & 0 \\ 0 & e^{\lambda_2 t} \end{bmatrix}.$$

This example demonstrates that, given any $\mathbf{A}$, the advantage of *diagonalizing* $\mathbf{A}$, i.e., pick-

ing a *nonsingular* matrix $\mathbf{P}$ such that $\tilde{\mathbf{A}} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}$ is diagonal; this is an example of *similarity transformation*, which is often done to ease analysis — more about this in Sect. 5.1.

Advanced: Frobenius' theorem

Example 6 also demonstrates Frobenius' theorem [HJS08, p. 78]: If a matrix $\mathbf{A}$ has eigenvalues λ_i, and if a function f is analytic in a region in the complex plane containing λ_i, then the matrix $f(\mathbf{A})$ has eigenvalues $f(\lambda_i)$. Since the eigenvalues of a diagonal matrix $\mathbf{A}t = \text{diag}(\lambda_1 t, \ldots, \lambda_n t)$ are exactly $\lambda_1 t, \ldots, \lambda_n t$, and since the function $f(z) = e^z$, where $z \in \mathbb{C}$, is analytic in any finite region of $\mathbb{C}$, the eigenvalues of $e^{\mathbf{A}t}$ are exactly $e^{\lambda_1 t}, \ldots, e^{\lambda_n t}$. We will study eigenvalues, which play the role of system poles, in detail in the next lecture.

Figure 4: The system matrix of an LTI system must be Hurwitz — not Hewitt — for the system to be asymptotically stable. Image from Wikipedia.

The most important information embedded in the matrix exponential is *stability*. If the system matrix has only eigenvalues with a negative real part, then the matrix exponential and hence the free response will decay over time, and we say the system matrix is *Hurwitz* or the system matrix is a *stability matrix*. Moreover, we say the system — more precisely, the *equilibrium state* — is *asymptotically stable*.

5 Realization

Define the Laplace transforms

$$\boldsymbol{Y}(s) \overset{\text{def}}{=} \mathcal{L}\{\boldsymbol{y}(t)\} = \begin{bmatrix} \mathcal{L}\{y_1(t)\} \\ \vdots \\ \mathcal{L}\{y_p(t)\} \end{bmatrix}, \quad \boldsymbol{U}(s) \overset{\text{def}}{=} \mathcal{L}\{\boldsymbol{u}(t)\} = \begin{bmatrix} \mathcal{L}\{u_1(t)\} \\ \vdots \\ \mathcal{L}\{u_m(t)\} \end{bmatrix},$$

then the *transfer function matrix* (or *transfer matrix* in short) is defined as $\mathbf{G}(s)$ such that

$$\boldsymbol{Y}(s) = \mathbf{G}(s)\boldsymbol{U}(s). \tag{15}$$

The transfer function matrix is related to $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ through

$$\mathbf{G}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}. \tag{16}$$

- The inverse matrix $(s\mathbf{I}-\mathbf{A})^{-1}$ is called the *resolvent matrix*, because it is the Laplace transform of the state transition matrix $e^{\mathbf{A}t}$. Notice when $s = \lambda$, where λ is any eigenvalue of $\mathbf{A}$, then by definition $\det(s\mathbf{I} - \mathbf{A}) = 0$, and the resolvent matrix does not exist. This is why the eigenvalues of $\mathbf{A}$ are also called the *poles*.
- The tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ is called a *realization* of $\mathbf{G}(s)$.

- While a transfer function matrix is unique, a realization is not unique, because if $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ is a realization, we can pick any nonsingular matrix $\mathbf{P}$ such that $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \mathbf{D})$ is also a realization, where

$$\tilde{\mathbf{A}} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}, \quad \tilde{\mathbf{B}} = \mathbf{P}^{-1}\mathbf{B}, \quad \tilde{\mathbf{C}} = \mathbf{C}\mathbf{P}. \tag{17}$$

  The realizations $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ and $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \mathbf{D})$ are so-called *similar*.

Quiz 5

If the state corresponding to realization $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ is $\boldsymbol{x}$, what is the state corresponding to the similar realization $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \mathbf{D})$?

The most important realization is the *minimal realization* (or minimum realization), which is a realization of the lowest order. Minimal realizations are controllable *and* observable models — the concept of *controllability* is a topic of the next lecture, whereas *observability* will have to wait until the next course. Nevertheless, a transfer matrix may or may not be *realizable*.

A transfer matrix $\mathbf{G}(s)$ is realizable if and only if $\mathbf{G}(s)$ is a *proper rational* matrix.

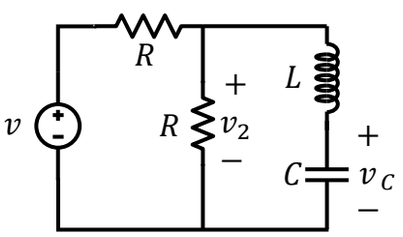

Figure 5: An RLC circuit with input v and output v_C.

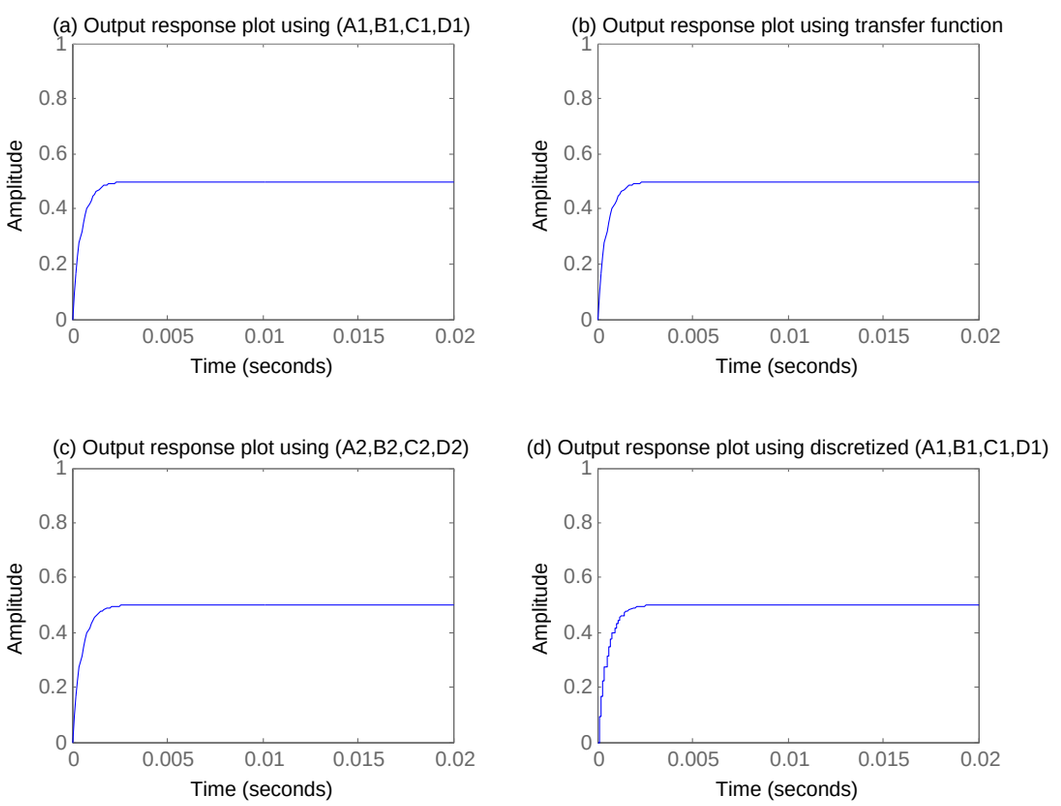

Figure 6: Unit step responses for Example 7.

Example 7

For the RLC circuit in Figure 5, we have derived the continuous-time state-space equations

in the tutorial as follows:

$$\begin{bmatrix} \ddot{v}_C \\ \dot{v}_C \end{bmatrix} = \begin{bmatrix} -\frac{R}{2L} & -\frac{1}{LC} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{v}_C \\ v_C \end{bmatrix} + \begin{bmatrix} \frac{1}{2LC} \\ 0 \end{bmatrix} v, \tag{18}$$

$$v_C = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{v}_C \\ v_C \end{bmatrix}. \tag{19}$$

Applying Eq. (16) gives us the transfer function:

$$\begin{aligned} G(s) &= \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} = \begin{bmatrix} 0 & 1 \end{bmatrix} \left(\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} -\frac{R}{2L} & -\frac{1}{LC} \\ 1 & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} \frac{1}{2LC} \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} s + \frac{R}{2L} & \frac{1}{LC} \\ -1 & s \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{2LC} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix} \frac{1}{s^2 + \frac{R}{2L}s + \frac{1}{LC}} \begin{bmatrix} s & -\frac{1}{LC} \\ 1 & s + \frac{R}{2L} \end{bmatrix} \begin{bmatrix} \frac{1}{2LC} \\ 0 \end{bmatrix} \\ &= \frac{1}{s^2 + \frac{R}{2L}s + \frac{1}{LC}} \begin{bmatrix} 1 & s + \frac{R}{2L} \end{bmatrix} \begin{bmatrix} \frac{1}{2LC} \\ 0 \end{bmatrix} = \frac{1}{s^2 + \frac{R}{2L}s + \frac{1}{LC}} \cdot \frac{1}{2LC} \\ &= \frac{1}{2LCs^2 + RCs + 2}. \end{aligned}$$

MATLAB provides the functions `ss2tf` and `tf2ss` for *numerical* conversions between state-space and transfer-matrix representations, but there are no MATLAB functions for *symbolic* conversions. Suppose $R = 1000$, $L = 10^{-6}$ and $C = 10^{-6}$, then we can use the following MATLAB code for conversion:

```
R=1000; L=1e-6; c=1e-6;
A1 = [-R/(2*L) -1/(L*c); 1 0]; B1 = [1/(2*L*c); 0]; C1 = [0 1]; D1 = 0;

[b,a] = ss2tf(A1,B1,C1,D1);
G = tf(b,a)
[A2,B2,C2,D2] = tf2ss(b,a)
```

In general, the matrices `A1`, `B1`, `C1`, `D1` are not the same as `A2`, `B2`, `C2`, `D2` in the code above. The unit step responses using the original state-space representation (`A1`, `B1`, `C1`, `D1`), the transfer function, and the realization (`A2`, `B2`, `C2`, `D2`) are given in Figure 6(a)-(c), which are identical.

5.1 Similarity transformation

Every system has a unique transfer-matrix representation but an infinite number of *equivalent* state-space representations, which can be transformed to one another through *similarity transformation*. But first, let us understand what similar matrices mean.

Pre-multiplying matrix $\mathbf{A}$ with a nonsingular matrix has the same effect as performing elementary row operations on $\mathbf{A}$. Similarly, post-multiplying $\mathbf{A}$ with a nonsingular matrix has the same effect as performing elementary column operations on $\mathbf{A}$.

Definition(s): Equivalent matrices [Mey01, p. 134], similar matrices [Mey01, p. 255]

Whenever matrix $\mathbf{B}$ can be derived from matrix $\mathbf{A}$ through a combination of elementary row and column operations, we say $\mathbf{A}$ and $\mathbf{B}$ are equivalent matrices, and write $\mathbf{A} \sim \mathbf{B}$. Since elementary row and column operations can be achieved through pre- and post-

multiplication by nonsingular matrices respectively, we conclude

$$\mathbf{A} \sim \mathbf{B} \iff \mathbf{PAQ} = \mathbf{B} \text{ for nonsingular } \mathbf{P} \text{ and } \mathbf{Q}.$$

If $\mathbf{A}$, $\mathbf{B}$ are square, and $\mathbf{P}$, $\mathbf{Q}$ are inverses of each other, we say $\mathbf{A}$ and $\mathbf{B}$ are similar matrices, and write $\mathbf{A} \simeq \mathbf{B}$.

We will now relate similar matrices to similarity transformation of state-space representations.

Given continuous-time LTI state-space equations

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t), \\ \boldsymbol{y}(t) &= \mathbf{C}\boldsymbol{x}(t) + \mathbf{D}\boldsymbol{u}(t), \end{aligned} \tag{20}$$

and a nonsingular matrix $\mathbf{T}$, we define

$$\tilde{\boldsymbol{x}}(t) = \mathbf{T}^{-1}\boldsymbol{x}(t),\ \tilde{\mathbf{A}} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T},\ \tilde{\mathbf{B}} = \mathbf{T}^{-1}\mathbf{B},\ \tilde{\mathbf{C}} = \mathbf{C}\mathbf{T},\ \tilde{\mathbf{D}} = \mathbf{D},$$

such that $\tilde{\mathbf{A}} \sim \mathbf{A}, \tilde{\mathbf{B}} \sim \mathbf{B}, \tilde{\mathbf{C}} \sim \mathbf{C}, \tilde{\mathbf{D}} \sim \mathbf{D}$. Then,

$$\begin{aligned} \dot{\tilde{\boldsymbol{x}}}(t) &= \tilde{\mathbf{A}}\tilde{\boldsymbol{x}}(t) + \tilde{\mathbf{B}}\boldsymbol{u}(t), \\ \boldsymbol{y}(t) &= \tilde{\mathbf{C}}\tilde{\boldsymbol{x}}(t) + \tilde{\mathbf{D}}\boldsymbol{u}(t), \end{aligned} \tag{21}$$

and we write $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}}) \sim (\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$.

The question now is what form of $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}})$ is "useful"? We have the following useful forms:

- The *controllable canonical form*, *observable canonical form*, and *Jordan canonical form* have historically been useful for analysis and proving theorems, but we shall not be concerned with them in this course.
- The *standard form for uncontrollable systems* can be obtained through the similarity transformation called *controllable subspace decomposition*. This is covered in the next lecture.
- Minimal realizations were mentioned before, and they can be obtained through the similarity transformation called *Kalman decomposition* (MATLAB command `minreal`), which decomposes a model into four parts: controllable and observable, controllable but unobservable, uncontrollable but observable, uncontrollable and unobservable. The controllable and observable part is exactly the minimal realization. Kalman decomposition is covered in the next course.

6 Discretizing continuous-time state-space equations

The solution for the continuous-time LTI state-space equations in Sect. 4 gives us the tool to discretize a continuous-time model. The state of a continuous-time LTI system with initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$ is given by

$$\boldsymbol{x}(t) = e^{\mathbf{A}t}\boldsymbol{x}_0 + e^{\mathbf{A}t}\int_0^t e^{-\mathbf{A}\tau}\mathbf{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau.$$

The state at $t = kT$, where T is the sampling time interval, is given by

$$\boldsymbol{x}(kT) = e^{\mathbf{A}kT}\boldsymbol{x}_0 + e^{\mathbf{A}kT}\int_0^{kT} e^{-\mathbf{A}\tau}\mathbf{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau. \tag{22}$$

The state at $t = (k+1)T$ is given by

$$\boldsymbol{x}((k+1)T) = e^{\mathbf{A}(k+1)T}\boldsymbol{x}_0 + e^{\mathbf{A}(k+1)T}\int_0^{(k+1)T} e^{-\mathbf{A}\tau}\mathbf{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau. \tag{23}$$

Our goal is to combine (22) and (23) to express $\boldsymbol{x}((k+1)T)$ as a linear combination of $\boldsymbol{x}(kT)$ and $\boldsymbol{u}(kT)$. For this, we need to get rid of the constant terms consisting of $\boldsymbol{x}_0$. Premultiplying Eq. (22) with $e^{\mathbf{A}T}$ gives us

$$e^{\mathbf{A}T}\boldsymbol{x}(kT) = e^{\mathbf{A}(k+1)T}\boldsymbol{x}_0 + e^{\mathbf{A}(k+1)T}\int_0^{kT} e^{-\mathbf{A}\tau}\mathbf{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau.$$

Subtracting the above from Eq. (23) produces

$$\boldsymbol{x}((k+1)T) = e^{\mathbf{A}T}\boldsymbol{x}(kT) + e^{\mathbf{A}(k+1)T}\int_{kT}^{(k+1)T} e^{-\mathbf{A}\tau}\mathbf{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau. \tag{24}$$

The equation above establishes the link between the state of the next instant and the state of the current instant through $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{u}(t)$, but the integral term can be further simplified. Through change of variable $\zeta \overset{\text{def}}{=} (k+1)T - \tau$ for simplifying the integration upper limit, we get

$$\begin{aligned}\boldsymbol{x}((k+1)T) &= e^{\mathbf{A}T}\boldsymbol{x}(kT) + e^{\mathbf{A}(k+1)T}\int_T^0 e^{-\mathbf{A}[(k+1)T-\zeta]}\mathbf{B}\boldsymbol{u}((k+1)T-\zeta)\,\mathrm{d}(-\zeta)\\ &= e^{\mathbf{A}T}\boldsymbol{x}(kT) + \int_0^T e^{\mathbf{A}\zeta}\mathbf{B}\boldsymbol{u}((k+1)T-\zeta)\,\mathrm{d}\zeta.\end{aligned}$$

Holding $\boldsymbol{u}((k+1)T-\zeta)$ constant over the interval $\zeta \in (0, T]$ means $\boldsymbol{u}((k+1)T-\zeta) = \boldsymbol{u}(kT)$ over the interval $\zeta \in (0, T]$ (this is the so-called *zero-order hold*), and

$$\boldsymbol{x}((k+1)T) = e^{\mathbf{A}T}\boldsymbol{x}(kT) + \int_0^T e^{\mathbf{A}\zeta}\mathbf{B}\,\mathrm{d}\zeta \cdot \boldsymbol{u}(kT).$$

At this point, we have successfully expressed $\boldsymbol{x}((k+1)T)$ as a linear combination of $\boldsymbol{x}(kT)$ and $\boldsymbol{u}(kT)$. By convention for discrete-time systems, we replace "(kT)" with "$[k]$", and write

$$\boldsymbol{x}[k+1] = e^{\mathbf{A}T}\boldsymbol{x}[k] + \int_0^T e^{\mathbf{A}\zeta}\mathbf{B}\,\mathrm{d}\zeta \cdot \boldsymbol{u}[k].$$

Given the continuous-time LTI state-space equations

$$\begin{aligned}\dot{\boldsymbol{x}}(t) &= \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t),\\ \boldsymbol{y}(t) &= \mathbf{C}\boldsymbol{x}(t) + \mathbf{D}\boldsymbol{u}(t),\end{aligned}$$

the discrete-time equivalent state-space equations are

$$\begin{aligned}\boldsymbol{x}[k+1] &= \mathbf{F}\boldsymbol{x}[k] + \mathbf{G}\boldsymbol{u}[k],\\ \boldsymbol{y}[k] &= \mathbf{C}\boldsymbol{x}[k] + \mathbf{D}\boldsymbol{u}[k].\end{aligned}$$

where

$$\mathbf{F} = e^{\mathbf{A}T}, \qquad \mathbf{G} = \int_0^T e^{\mathbf{A}\zeta}\mathbf{B}\,\mathrm{d}\zeta.$$

Since our derivation above is based on the assumption that all signals change values in a step-wise manner, the procedure is sometimes called the *step-invariant transformation*.

The step-invariant transformation tells us that if a continuous-time system has system matrix $\mathbf{A}$, the discrete-time counterpart has system matrix $e^{\mathbf{A}T}$.

According to Frobenius' theorem, if $\mathbf{A}$ has eigenvalues λ_i, then $e^{\mathbf{A}T}$ has eigenvalues $e^{\lambda_i T}$.

This fact will be useful for controller design later.

Example 8

Revisiting the RLC circuit in Example 7, it is possible to discretize the state-space equations (18)-(19) *symbolically*, but the algebraic expression is rather lengthy. Instead, suppose again $R = 1000$, $L = 10^{-6}$ and $C = 10^{-6}$, then the following MATLAB code returns the discretized model:

```
T = 1e-4;
sysd = c2d(sys, T, 'zoh');
```

Figure 6(d) shows the discrete-time unit step response of the RLC circuit based on the discretized model.

6.1 Solving discrete-time state-space equations

Equipped with a way of solving LTI state-space equations, and a way of discretizing continuous-time LTI state-space equations, the natural next step is to find a way to solve discrete-time LTI state-space equations.

Given the discrete-time system

$$\boldsymbol{x}[k+1] = \mathbf{F}\boldsymbol{x}[k] + \mathbf{G}\boldsymbol{u}[k], \quad \boldsymbol{y}[k] = \mathbf{C}\boldsymbol{x}[k] + \mathbf{D}\boldsymbol{u}[k], \quad \boldsymbol{x}[0] = \boldsymbol{x}_0,$$

we can see that

$$\begin{aligned}
\boldsymbol{x}[1] &= \mathbf{F}\boldsymbol{x}_0 + \mathbf{G}\boldsymbol{u}[0], \\
\boldsymbol{x}[2] &= \mathbf{F}(\mathbf{F}\boldsymbol{x}_0 + \mathbf{G}\boldsymbol{u}[0]) + \mathbf{G}\boldsymbol{u}[1] = \mathbf{F}^2\boldsymbol{x}_0 + \mathbf{F}\mathbf{G}\boldsymbol{u}[0] + +\mathbf{G}\boldsymbol{u}[1], \\
&\vdots \\
\boldsymbol{x}[k] &= \mathbf{F}^k\boldsymbol{x}_0 + \sum_{i=0}^{k-1} \mathbf{F}^{k-1-i}\mathbf{G}\boldsymbol{u}[i].
\end{aligned}$$

Therefore, the solution of the discrete-time state-space equations is

$$\boldsymbol{x}[k] = \mathbf{F}^k\boldsymbol{x}_0 + \sum_{i=0}^{k-1} \mathbf{F}^{k-1-i}\mathbf{G}\boldsymbol{u}[i], \tag{25}$$

$$\boldsymbol{y}[k] = \mathbf{C}\mathbf{F}^k\boldsymbol{x}_0 + \sum_{i=0}^{k-1} \mathbf{C}\mathbf{F}^{k-1-i}\mathbf{G}\boldsymbol{u}[i] + \mathbf{D}\boldsymbol{u}[k]. \tag{26}$$

Eq. (25) shows that the state transition matrix of a discrete-time system is the matrix power $\mathbf{F}^k$.

The most important information embedded in the matrix power is *stability*. If the system matrix has only eigenvalues with a magnitude less than 1, then the matrix power and hence the free response will decay over time, and we say the system matrix is *convergent*/*discrete-Hurwitz* or the system matrix is a *stability matrix*. Moreover, we say the system — more precisely, the *equilibrium state* — is *asymptotically stable*.

References

[Cor04] J.-P. Corriou, *Process Control: Theory and Applications*, Springer-Verlag London Ltd., 2004.

[DAP88] E. Duke, R. Antoniewicz, and B. Patterson, Derivation and definition of a linear aircraft model, *NASA Reference Publication* **1207** (1988).

[HJS08] E. Hendricks, O. Jannerup, and P. H. Sorensen, *Linear Systems Control: Deterministic and Stochastic Methods*, Springer-Verlag Berlin Heidelberg, 2008.

[Mey01] C. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, 2001.

[ML03] C. Moler and C. V. Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, *SIAM Review* **45** no. 1 (2003), 3–49. https://doi.org/10.1137/S00361445024180.

[WL07] R. L. Williams II and D. A. Lawrence, *Linear State-Space Control Systems*, John Wiley & Sons, 2007.

[Żak03] S. H. Żak, *Systems and Control*, Oxford University Press, 2003.