Note 10: Discretization, Disturbances, System ID, Verification

1 Overview and Motivation

Recall that in Note 7 we introduced two types of models: the *discrete-time LTI difference equation model* and the *continuous-time LTI differential equation model*. We also solved for the state trajectory of each model.

Sadly, it is non-trivial to actually do control in practice using just the idealized framework we have built so far. There are four main reasons why this is the case, and finding ways to deal with them is the focus of this note.

1. Suppose we know that our system is accurately modeled by a continuous-time LTI differential equation model. Our computers live in discrete time, and so they cannot completely simulate continuous-time models. If we want to implement a controller on a computer, then we must use instead a discrete-time model that closely approximates the continuous-time model.

   > **Key Idea 1** (Discretization)
   > *Discretization* is the technique of finding a discrete-time model that approximates a given continuous-time model.

2. Suppose we build or observe a physical system that we would like to model and do control on. Even if we suspect that the system is modeled by, say, a discrete-time LTI difference equation model, there are probably factors in the environment that we didn't account for in our simple linear model. We need to learn how to deal with these factors systematically.

   > **Key Idea 2** (Disturbance)
   > *Disturbances* are factors in the underlying system that are not accounted for in our model.

3. Suppose we build or observe a physical system that we would like to model and do control on. Even if we suspect that our system is modeled by, say, a discrete-time LTI difference equation model, we still need to find the A and B matrices (i.e., the *model parameters*), in order to do anything useful with the model.

   > **Key Idea 3** (System Identification)
   > *System identification* is the technique of learning estimates for the model parameters from data.

4. Suppose we have built a discrete-time model for our system and estimated the parameters. This is great, but we don't know if our model actually works. To ensure that our model is accurate, we would like to verify its performance in some way.

> **Key Idea 4** (Validation)
> *Validation* is the technique of "checking our work" – quantifying the accuracy of our model compared with reality, by testing the performance of our model on data that wasn't used to construct it.

Once we have these four ideas under our belt, we are able to spin up a model for a given physical system and use it to make predictions that we are reasonably confident about. This opens up avenues to accurately manipulate the system via control inputs in various ways that we will learn about in subsequent notes. In essence, this opens the gateway to *practical control*.

Although the principles we will discuss generally apply to all control models, we will concretize them by applying them to models we have seen before, namely the models in Note 7:

Model 5 (Continuous-Time LTI Differential Equation Model)
The model is of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}_c(t) = A_c\vec{x}_c(t) + B_c\vec{u}_c(t) \qquad t \in \mathbb{R}_+ \tag{1}$$

$$\vec{x}_c(0) = \vec{x}_0 \tag{2}$$

where $\vec{x}_c\colon \mathbb{R}_+ \to \mathbb{R}^n$ is the state of the system as a function of time, $\vec{u}_c\colon \mathbb{R}_+ \to \mathbb{R}^m$ is the control input as a function of time, and $A_c \in \mathbb{R}^{n\times n}$, $B_c \in \mathbb{R}^{n\times m}$ are matrices.

Model 6 (Discrete-Time LTI Difference Equation Model)
The model is of the form

$$\vec{x}_d[i+1] = A_d\vec{x}_d[i] + B_d\vec{u}_d[i] \qquad i \in \mathbb{N} \tag{3}$$

$$\vec{x}_d[0] = \vec{x}_0 \tag{4}$$

where $\vec{x}_d\colon \mathbb{N} \to \mathbb{R}^n$ is the state of the system as a function of timestep, $\vec{u}_d\colon \mathbb{N} \to \mathbb{R}^m$ is the control input as a function of timestep, and $A_d \in \mathbb{R}^{n\times n}$, $B_d \in \mathbb{R}^{n\times m}$ are matrices.

NOTE: We use the c and d subscripts to distinguish continuous and discrete time.

2 Discretization

We want to *approximate* a continuous-time LTI differential equation model by a discrete-time LTI difference equation model.

2.1 Tracing a Continuous-Time System

One way we can start this approximation process is to let our $\vec{x}_d[i]$ be *traces*, e.g., samples of our system state $\vec{x}_c$ at discrete timesteps of length Δ. This means that we will be working with the quantities $\vec{x}_d[0] = \vec{x}_c(0)$, $\vec{x}_d[1] = \vec{x}_c(\Delta)$, $\vec{x}_d[2] = \vec{x}_c(2\Delta)$, etc.. In general, we use the traced states

$$\vec{x}_d[i] := \vec{x}_c(i\Delta) \qquad i \in \mathbb{N}. \tag{5}$$

Similarly, we use the traced inputs

$$\vec{u}_d[i] := \vec{u}_c(i\Delta) \qquad i \in \mathbb{N}. \tag{6}$$

Now, using these traced states and inputs, and knowing that our system evolves according to Continuous-Time LTI Differential Equation Model, we would like to make a Discrete-Time LTI Difference Equation Model. More formally, knowing that our system obeys the differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}_c(t) = A_c\vec{x}_c(t) + B_c\vec{u}_c(t) \qquad t \in \mathbb{R}_+ \tag{7}$$

we would like to find $A_d \in \mathbb{R}^{n\times n}$ and $B_d \in \mathbb{R}^{n\times m}$, independent of the timestep i, the initial condition $\vec{x}_0$, and the inputs $\vec{u}_c$, such that

$$\vec{x}_d[i+1] = A_d\vec{x}_d[i] + B_d\vec{u}_d[i] \qquad i \in \mathbb{N} \tag{8}$$

$$\text{or more explicitly} \qquad \vec{x}_c((i+1)\Delta) = A_d\vec{x}_c(i\Delta) + B_d\vec{u}_c(i\Delta) \qquad i \in \mathbb{N}. \tag{9}$$

From now on the equivalence of the two above notations is taken for granted.

2.2 Piecewise Constant Input Assumption

Unfortunately, the desire we have is flat-out not possible to achieve, no matter how clever we are in our analysis, unless we have more information about the control inputs $\vec{u}_c$ between the endpoints $i\Delta$. The reason is that we don't have any information about how $\vec{u}_c$ behaves in between the traces $\vec{u}_d[i]$ and $\vec{u}_d[i+1]$ – it can be arbitrarily badly behaved within the interval $(i\Delta, (i+1)\Delta)$. So, it would be impossible to get coefficients A_d and B_d which multiply $\vec{x}_d[i]$ and $\vec{u}_d[i]$ to get $\vec{x}_d[i+1]$ that work for any input $\vec{u}_c$.

To fix this, we will make the *reasonable* assumption that $\vec{u}_c$ is piecewise constant. More formally, we have

Assumption 7 (Piecewise Constant Inputs)
The inputs $\vec{u}_c$ are piecewise constant on intervals of width Δ, that is,

$$\vec{u}_c(t) = \vec{u}_c(i\Delta) = \vec{u}_d[i] \qquad \text{for all } t \in [i\Delta, (i+1)\Delta). \tag{10}$$

NOTE: This assumption is also called a *zero-order hold* assumption. This is because the input is piecewise a zero-order polynomial in time, i.e., is piecewise constant.

At first glance, this assumption may seem *unreasonable*. For example, if we are studying a circuit and our input is some form of AC current (and hence sinusoidal), then there is *no* time interval on which the input is constant. The point is that if the input $\vec{u}_c$ is continuous[1], then the discretized function $\vec{u}_d$ approximates $\vec{u}_c$ for small Δ. In math,

$$\vec{u}_c(t) = \vec{u}_c\left(\Delta \cdot \frac{t}{\Delta}\right) \approx \vec{u}_c\left(\Delta \left\lfloor \frac{t}{\Delta} \right\rfloor\right) = \vec{u}_d\left[\left\lfloor \frac{t}{\Delta} \right\rfloor\right]. \tag{11}$$

The approximation further becomes exact in the limit $\Delta \to 0$, but we need more mathematical analysis tools to show this. For now, we may take it as given. A pictorial representation of what is going on for $\vec{u}_c(t) = \sin(t)$ is below:

[1] We may relax this assumption to right-continuous.

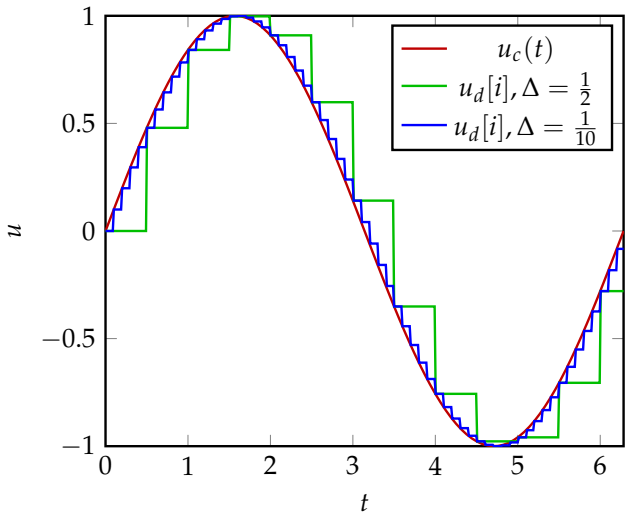

Figure 1: An example of a piecewise constant function where the limit as the time-step Δ goes to 0 approaches a continuous function. The red line, the original signal $u_c(t) = \sin(t)$, is traced almost exactly by the blue line (small Δ) and not nearly as well by the green line (large Δ).

Whether by the formalism or by the diagram, it should now seem *reasonable* that when Δ is tiny, a piecewise constant approximation for $\vec{u}_c$ can be a very accurate approximation of $\vec{u}_c$. So, we may as well assume that $\vec{u}_c$ is piecewise constant. This is how we justify the piecewise constant assumption. The new model is:

Model 8 (Continuous-Time LTI Differential Equation Model with Piecewise Constant Inputs)
The model is of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}_c(t) = A_c\vec{x}_c(t) + B_c\vec{u}_d[i] \qquad t \in [i\Delta, (i+1)\Delta) \qquad i \in \mathbb{N} \tag{12}$$

$$\vec{x}_c(0) = \vec{x}_0 \tag{13}$$

where $\vec{x}_c \colon \mathbb{R}_+ \to \mathbb{R}^n$ is the state of the system as a function of time, $\vec{u}_d \colon \mathbb{N} \to \mathbb{R}^m$ is the traced control input as a function of timestep, and $A_c \in \mathbb{R}^{n\times n}$, $B_c \in \mathbb{R}^{n\times m}$ are matrices.

2.3 Calculating Discretization Coefficients

We return to the problem of discretization, now with the piecewise constant input assumption.

Notice that in this section we begin with the diagonalizable case, whereas in Note 7 we built up to the diagonalizable case by way of the scalar case and diagonal case. This is because, unlike in Note 7, the proofs for all versions of this theorem are essentially identical, just using the corresponding results from Note 7. Hence we lose almost no insight by looking at the most general case.

Theorem 9 (Discretization of Continuous-Time LTI Differential Equation Model with Piecewise Constant Inputs When A_c is Diagonalizable and Invertible)
Suppose in the Continuous-Time LTI Differential Equation Model with Piecewise Constant Inputs that A_c is diagonalizable with diagonalization $A_c = V\Lambda V^{-1}$, and also invertible. The discretization of this model by a Discrete-Time LTI Difference Equation Model has coefficients

$$A_d = Ve^{\Lambda\Delta}V^{-1} \tag{14}$$
$$B_d = V(e^{\Lambda\Delta} - I_n)\Lambda^{-1}V^{-1}B_c \tag{15}$$

using the exponential notation from Note 9, i.e., if $\Lambda = \text{diag}(\lambda_{11}, \lambda_{22}, \ldots, \lambda_{nn})$ then $e^{\Lambda\Delta} = \text{diag}\left(e^{\lambda_{11}\Delta}, e^{\lambda_{22}\Delta}, \ldots, e^{\lambda_{nn}\Delta}\right)$.

Proof. The proof is in several steps, as is very common for longer mathematical proofs. We proceed methodically by changes-of-variables and reducing to problems we have already solved, e.g., in Note 9.

Step 1. We first look at a particular timestep i. Namely, since we are looking for an update equation which writes $\vec{x}_d[i+1]$ in terms of $\vec{x}_d[i]$ and $\vec{u}_d[i]$, we may assume that our differential equation starts at timestep i and goes until timestep $i+1$, and we know everything up to timestep i including $\vec{x}_d[i]$ and $\vec{u}_d[i]$. This gives

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}_c(t) = A_c\vec{x}_c(t) + B_c\vec{u}_d[i] \qquad t \in [i\Delta, (i+1)\Delta) \qquad \text{for this specific } i \tag{16}$$
$$\vec{x}_c(i\Delta) = \vec{x}_d[i] \qquad \text{for this specific } i. \tag{17}$$

Step 2. Now we solve the differential equation by reducing it to a previous differential equation we have already solved. Recall that in Note 7 we solved differential equations of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) = A\vec{x}(t) + B\vec{u}(t) \tag{18}$$
$$\vec{x}(0) = \vec{x}_0. \tag{19}$$

There are two differences between the differential equations:

- The notation is different – $\vec{x}_c$ instead of $\vec{x}$, $\vec{u}_c$ instead of $\vec{u}$, etc., and the initial condition is $\vec{x}_d[i]$.
- The initial condition is at time $t = i\Delta$ instead of $t = 0$.
- The differential equation we have to solve is only valid in an interval – the equation we have already solved is valid everywhere.

The second is the only essential difference, since the first is just a relabeling, and the third is solved by declaring that our solution is only valid in the particular interval. To get around the second difference, we will use *change of variables*. More specifically, let s be defined by

$$s := t - i\Delta \tag{20}$$

so that if $t \in [i\Delta, (i+1)\Delta)$ then $s \in [0, \Delta)$; and let $\vec{\tilde{x}}_c$ be defined pointwise on $[0, \Delta)$ by

$$\vec{\tilde{x}}_c(s) := \vec{x}_c(s + i\Delta) = \vec{x}_c(t). \tag{21}$$

Then

$$\vec{\tilde{x}}_c(0) = \vec{x}_c(i\Delta) = \vec{x}_d[i]. \tag{22}$$

The differential equation in these coordinates becomes

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}s}\vec{\tilde{x}}_c(s) &= \frac{\mathrm{d}}{\mathrm{d}s}\vec{x}_c(t) && (23)\\
&= \underbrace{\frac{\mathrm{d}t}{\mathrm{d}s}}_{=1} \cdot \frac{\mathrm{d}\vec{x}_c(t)}{\mathrm{d}t} && (24)\\
&= A_c\vec{x}_c(t) + B_c\vec{u}_c(t) && (25)\\
&= A_c\vec{\tilde{x}}_c(s) + B_c\vec{u}_d[i] \qquad s \in [0,\Delta). && (26)
\end{aligned}$$

So, we have to solve the problem

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}s}\vec{\tilde{x}}_c(s) &= A_c\vec{\tilde{x}}_c(s) + B_c\vec{u}_d[i] \qquad s \in [0,\Delta) && (27)\\
\vec{\tilde{x}}_c(0) &= \vec{x}_d[i]. && (28)
\end{aligned}$$

By Note 7 the solution is

$$\vec{\tilde{x}}_c(s) = V\mathrm{e}^{\Lambda s}V^{-1}\vec{x}_d[i] + \int_0^s V\mathrm{e}^{\Lambda(s-\tau)}V^{-1}B_c\vec{u}_d[i]\,\mathrm{d}\tau \qquad s \in [0,\Delta). \tag{29}$$

Simplifying and using the fact that integrals are linear, we get

$$\vec{\tilde{x}}_c(s) = V\mathrm{e}^{\Lambda s}V^{-1}\vec{x}_d[i] + V\mathrm{e}^{\Lambda s}\left(\int_0^s \mathrm{e}^{-\Lambda\tau}\,\mathrm{d}\tau\right)V^{-1}B_c\vec{u}_d[i] \qquad s \in [0,\Delta). \tag{30}$$

To compute this integral, we note that Λ is diagonal and invertible, and hence we can write $\Lambda = \mathrm{diag}(\lambda_{11},\lambda_{22},\ldots,\lambda_{nn})$ with $\lambda_{ii} \neq 0$.[2] Thus

$$\begin{aligned}
\int_0^s \mathrm{e}^{-\Lambda\tau}\,\mathrm{d}\tau &= \int_0^s \begin{bmatrix} \mathrm{e}^{-\lambda_{11}\tau} & & & \\ & \mathrm{e}^{-\lambda_{22}\tau} & & \\ & & \ddots & \\ & & & \mathrm{e}^{-\lambda_{nn}\tau} \end{bmatrix} \mathrm{d}\tau && (31)\\
&= \begin{bmatrix} -\frac{1}{\lambda_{11}}\mathrm{e}^{-\lambda_{11}\tau} & & & \\ & -\frac{1}{\lambda_{22}}\mathrm{e}^{-\lambda_{22}\tau} & & \\ & & \ddots & \\ & & & -\frac{1}{\lambda_{nn}}\mathrm{e}^{-\lambda_{nn}\tau} \end{bmatrix}\Bigg|_{\tau=0}^{\tau=s} && (32)\\
&= \left(-\mathrm{e}^{-\Lambda\tau}\Lambda^{-1}\right)\Big|_{\tau=0}^{\tau=s} && (33)\\
&= \left(I_n - \mathrm{e}^{-\Lambda s}\right)\Lambda^{-1}. && (34)
\end{aligned}$$

Plugging the value of the integral back in, we have

$$\begin{aligned}
\vec{\tilde{x}}_c(s) &= V\mathrm{e}^{\Lambda s}V^{-1}\vec{x}_d[i] + V\mathrm{e}^{\Lambda s}\left(I_n - \mathrm{e}^{-\Lambda s}\right)\Lambda^{-1}V^{-1}B_c\vec{u}_d[i] \qquad s \in [0,\Delta) && (35)\\
&= V\mathrm{e}^{\Lambda s}V^{-1}\vec{x}_d[i] + V\left(\mathrm{e}^{\Lambda s} - I_n\right)\Lambda^{-1}V^{-1}B_c\vec{u}_d[i] \qquad s \in [0,\Delta). && (36)
\end{aligned}$$

[2] This is a side argument, but it may not be clear why we can have $\lambda_{ii} \neq 0$. This is because A_c is invertible, so A_c^{-1} exists, so $\Lambda^{-1} = V^{-1}A_c^{-1}V$ exists and Λ is invertible. Since Λ is diagonal, Λ^{-1} is diagonal, and its entries are $\frac{1}{\lambda_{ii}}$.

Step 3. Now we solve for $\vec{x}_d[i+1]$. Indeed,

$$\vec{x}_d[i+1] = \vec{x}_c((i+1)\Delta) \tag{37}$$
$$= \vec{\tilde{x}}_c(\Delta) \tag{38}$$
$$= Ve^{\Lambda\Delta}V^{-1}\vec{x}_d[i] + V\left(e^{\Lambda\Delta} - I_n\right)\Lambda^{-1}V^{-1}B_c\vec{u}_d[i]. \tag{39}$$

We see that this is a linear equation for $\vec{x}_d[i+1]$ in terms of $\vec{x}_d[i]$ in the form of eq. (3) with coefficients

$$A_d = Ve^{\Lambda\Delta}V^{-1} \qquad \text{and} \qquad B_d = V(e^{\Lambda\Delta} - I_n)\Lambda^{-1}V^{-1}B_c \tag{40}$$

as desired.

Concept Check: We used the assumption that Λ is invertible in Step 2, i.e., eq. (32), when we computed the integral $\int_0^s e^{-\Lambda\tau}\,d\tau$. You can compute this integral even when Λ has some 0 entries, i.e., A has some 0 eigenvalues and is thus not invertible. Do this computation – you won't get a nice closed form, unfortunately. Instead, the i^{th} diagonal entry of the solution to the integral will be calculated differently depending on whether $\lambda_{ii} = 0$ (in which case $e^{\lambda_{ii}\tau} = 1$) or $\lambda_{ii} \neq 0$ (in which case we use the computation above). Carry through the rest of the coefficient calculation again with your more general solution to the integral. What are the discretization coefficients in this case? This calculation allows us to drop the assumption that A_c is invertible, and really we only need that A_c is diagonalizable. □

Phew! That was a difficult and lengthy calculation. Now let's do some examples.

2.4 Examples

Suppose we have the scalar system

$$\frac{d}{dt}x_c(t) = 3x_c(t) + 5u_c(t) \tag{41}$$

where u_c is piecewise constant over intervals of length Δ.

In this case we use theorem 9 with $A_c = 3$ "diagonalized" using $V = 1$ and $\Lambda = 3$, as well as $B_c = 5$, to get the discretization

$$x_d[i+1] = e^{3\Delta}x_d[i] + \frac{e^{3\Delta} - 1}{3} \cdot 5u_d[i]. \tag{42}$$

Now suppose we have the diagonal system

$$\frac{d}{dt}\vec{x}_c(t) = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}\vec{x}_c(t) + \begin{bmatrix} 5 \\ 1 \end{bmatrix}u_c(t) \tag{43}$$

where u_c is again piecewise constant over intervals of length Δ.

In this case we use theorem 9 with $A_c = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$ "diagonalized" using $V = I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\Lambda = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$, as well as $B_c = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$, to get the discretization

$$x_d[i+1] = \begin{bmatrix} e^{3\Delta} & 0 \\ 0 & e^{2\Delta} \end{bmatrix}x_d[i] + \begin{bmatrix} \frac{e^{3\Delta}-1}{3} & 0 \\ 0 & \frac{e^{2\Delta}-1}{2} \end{bmatrix}\begin{bmatrix} 5 \\ 1 \end{bmatrix}u_d[i]. \tag{44}$$

We will conclude this section by discussing some generalizations and asymptotic analysis.

2.5 (OPTIONAL) Beyond Diagonalization

NOTE: This section has the section on *Matrix Exponentials* in Note 7 as a prerequisite.

If we are allowed to use the matrix exponential we can generalize the earlier analysis.

Theorem 10 (Discretization of Continuous-Time LTI Differential Equation Model with Piecewise Constant Inputs)
The discretization of Continuous-Time LTI Differential Equation Model with Piecewise Constant Inputs by a Discrete-Time LTI Difference Equation Model has coefficients

$$A_d = \mathrm{e}^{A_c\Delta} \tag{45}$$

$$B_d = \mathrm{e}^{A_c\Delta}\left(\sum_{i=1}^{\infty}\frac{(-A_c)^{i-1}\Delta^i}{i!}\right)B_c. \tag{46}$$

Concept Check: Prove theorem 10. The following computation will be useful.

$$\int_0^t \mathrm{e}^{-A_c\tau}\,\mathrm{d}\tau = \int_0^t \sum_{i=0}^{\infty}\frac{(-A_c\tau)^i}{i!}\,\mathrm{d}\tau = \sum_{i=0}^{\infty}\int_0^t \frac{(-A_c\tau)^i}{i!}\,\mathrm{d}\tau = \sum_{i=0}^{\infty}\int_0^t \frac{(-A_c)^i}{i!}\tau^i\,\mathrm{d}\tau \tag{47}$$

$$= \sum_{i=0}^{\infty}\frac{(-A_c)^i}{i!}\int_0^t \tau^i\,\mathrm{d}\tau = \sum_{i=0}^{\infty}\frac{(-A_c)^i}{(i+1)!}t^{i+1} = \sum_{i=1}^{\infty}\frac{(-A_c)^{i-1}}{i!}t^i. \tag{48}$$

Concept Check: Show that theorem 11 is a special case of theorem 10. Also show that theorem 9 is a special case of theorem 11.

Theorem 11 (Discretization of Continuous-Time LTI Differential Equation Model with Piecewise Constant Inputs When A_c is Invertible)
Suppose in the Continuous-Time LTI Differential Equation Model with Piecewise Constant Inputs that A_c is invertible. Then the discretization of the model by a Discrete-Time LTI Difference Equation Model has coefficients

$$A_d = \mathrm{e}^{A_c\Delta} \tag{49}$$

$$B_d = \left(\mathrm{e}^{A_c\Delta} - I_n\right)A_c^{-1}B_c. \tag{50}$$

Concept Check: Prove theorem 11. The following identity will be helpful:

$$\int_0^t \mathrm{e}^{-A_c\tau}\,\mathrm{d}\tau = -\mathrm{e}^{-A_c\tau}A_c^{-1}\Big|_{\tau=0}^{\tau=t} = (I_n - \mathrm{e}^{-A_ct})A_c^{-1}. \tag{51}$$

2.6 (OPTIONAL) Towards Euler Discretization

Since for our applications in control we usually want Δ to be small, it is instructive to see the behavior of the discretization coefficients in the limit $\Delta \to 0$. The way that we do this is to examine the discretization coefficients, given in theorem 10, and truncate their series expansions at the terms which are linear in Δ. This is justifiable because when $\Delta < 1$ is small, $\Delta^2 \ll \Delta$, $\Delta^3 \ll \Delta^2$, and so on; hence all terms which are second-order or higher in Δ become negligible and may be ignored.

Using the coefficients A_d and B_d from theorem 10, we see

$$A_d = \mathrm{e}^{A_c\Delta} = I_n + A_c\Delta + \frac{(A_c\Delta)^2}{2} + \cdots \approx I_n + A_c\Delta \tag{52}$$

$$B_d = e^{A_c\Delta}\left(\sum_{i=1}^{\infty}\frac{(-A_c)^{i-1}\Delta^i}{i!}\right)B_c = \left(\sum_{i=0}^{\infty}\frac{(A_c\Delta)^i}{i!}\right)\left(\sum_{i=1}^{\infty}\frac{(-A_c)^{i-1}\Delta^i}{i!}\right)B_c \tag{53}$$

$$= (I_n + A_c\Delta + \cdots)\left(\Delta - \frac{A_c\Delta^2}{2} + \cdots\right)B_c \approx B_c\Delta. \tag{54}$$

This implies that for Δ small enough, an approximate discretization gives a model of the form

$$\vec{x}_d[i+1] \approx (I_n + A_c\Delta)\vec{x}_d[i] + \Delta B_c\vec{u}_d[i] \qquad i \in \mathbb{N}. \tag{55}$$

A generalization of this process is called *Euler discretization* and is used to approximately solve nonlinear differential equations.

3 Disturbances and Error

Thus far, our analysis of Continuous-Time LTI Differential Equation Model and Discrete-Time LTI Difference Equation Model has assumed that the original model we is a perfect description of the system. In practice, this is not the case. The relevant concept to describe this deviation from the model is *disturbance*. We have to deal with *disturbances*, which incorporate random noise from the environment, as well as systemic error, caused by having models which are too simple to describe the inherently very complicated natural system.

Definition 12 (Disturbance)
A *disturbance* is any of the following sources of error in a model:

- random noise from the environment;
- approximation error that comes from having wrong models;
- a combination of the above.

More specifically, the Wikipedia page on uncertainty quantification gives a more thorough list of possible error sources, which we summarize here. *You do not need to memorize the vocabulary we present here* – but this list might be interesting or helpful for your own enrichment.

- *Parameter disturbance* – This comes from the model parameters which govern the state trajectory, but whose exact values are unknown in practice and cannot be perfectly estimated.
- *Parameteric disturbance* – This comes from the control inputs which also govern the state trajectory, but whose exact values may not be exactly the same as the specification or plan.
- *Structural disturbance* – This comes from the fact that the model is only an approximation to reality, and will never be perfect. In practice there is usually a tradeoff between making the model simple (and solvable) and making it more accurate to reality; the gap between reality and our simplified model is called structural disturbance.
- *Algorithmic disturbance* – This comes from the fact that our computer algorithms will have implementation-dependent errors, such as numerical imprecision and overflow.

- *Experimental disturbance* – This comes from the variability in measurements; it is inevitable, and manifests as random noise when repeating the same process many times.
- *Interpolation disturbance* – This comes from a possible lack of historical data which is similar to the input data for the current experiment, at which point the model must interpolate or extrapolate to give an accurate state trajectory.

In general a real-world model will have all of these sources of disturbance.
To explicitly quantify disturbances in our models, we usually use *additive disturbances*.

Model 13 (Continuous-Time LTI Differential Equation Model With Additive Disturbance)
The model is of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}_c(t) = A_c\vec{x}_c(t) + B_c\vec{u}_c(t) + \vec{w}_c(t) \qquad t \in \mathbb{R}_+ \tag{56}$$

$$\vec{x}_c(0) = \vec{x}_0 \tag{57}$$

where $\vec{x}_c\colon \mathbb{R}_+ \to \mathbb{R}^n$ is the state of the system as a function of time, $\vec{u}_c\colon \mathbb{R}_+ \to \mathbb{R}^m$ is the control input as a function of time, $\vec{w}_c\colon \mathbb{R}_+ \to \mathbb{R}^n$ is the disturbance in the system as a function of time, and $A_c \in \mathbb{R}^{n\times n}, B_c \in \mathbb{R}^{n\times m}$ are matrices.

Model 14 (Discrete-Time LTI Difference Equation Model With Additive Disturbance)
The model is of the form

$$\vec{x}_d[i+1] = A_d\vec{x}_d[i] + B_d\vec{u}_d[i] + \vec{w}_d[i] \qquad i \in \mathbb{N} \tag{58}$$

$$\vec{x}_d[0] = \vec{x}_0 \tag{59}$$

where $\vec{x}_d\colon \mathbb{N} \to \mathbb{R}^n$ is the state of the system as a function of timestep, $\vec{u}_d\colon \mathbb{N} \to \mathbb{R}^m$ is the control input as a function of timestep, $\vec{w}_d\colon \mathbb{N} \to \mathbb{R}^n$ is the disturbance in the system as a function of timestep, and $A_d \in \mathbb{R}^{n\times n}, B_d \in \mathbb{R}^{n\times m}$ are matrices.

There are generally two ways we deal with additive disturbances in this class:

- We may treat $\vec{w}$ as another input.
  - Starting with the Continuous-Time LTI Differential Equation Model With Additive Disturbance, we may define a new input $\vec{v}_c(t) := \begin{bmatrix}\vec{u}_c(t)\\ \vec{w}_c(t)\end{bmatrix}$ and a new matrix $G_c := \begin{bmatrix}B_c & I_n\end{bmatrix}$, so that the model becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}_c(t) = A_c\vec{x}_c(t) + G_c\vec{v}_c(t) \tag{60}$$

$$\vec{x}_c(0) = \vec{x}_0 \tag{61}$$

    which is a Continuous-Time LTI Differential Equation Model.
  - Starting with the Discrete-Time LTI Difference Equation Model With Additive Disturbance, we may define a new input $\vec{v}_d[i] := \begin{bmatrix}\vec{u}_d[i]\\ \vec{w}_d[i]\end{bmatrix}$ and a new matrix $G_d := \begin{bmatrix}B_d & I_n\end{bmatrix}$, so that the difference

equation becomes

$$\vec{x}_d[i+1] = A_d\vec{x}_d[i] + G_d\vec{v}_d[i] \tag{62}$$
$$\vec{x}_d[0] = \vec{x}_0 \tag{63}$$

which is a Discrete-Time LTI Difference Equation Model.

Conceptually, this says that noise is another input to the model, albeit one we cannot control. We will thus be able to quantitatively analyze the effects of disturbance using all the techniques we will have developed for our familiar disturbance-less Discrete-Time LTI Difference Equation Models and Continuous-Time LTI Differential Equation Models. Under this viewpoint, *disturbances are nothing but inputs to the model – albeit not ones that the controller provides*.

- We may assume the disturbances are negligible. In particular, in many cases it is plausible that disturbances are small, and have a small effect on the system. Thus we may remove the $\vec{w}$ term from our calculations entirely – in effect setting $\vec{w} = \vec{0}_n$ – at the cost of turning the equal signs $=$ in the Continuous-Time LTI Differential Equation Model and Discrete-Time LTI Difference Equation Model into approximations $\approx$.

The upshot of both of these methods is the following.

Key Idea 15

We may develop our theoretical ideas while ignoring disturbance, i.e., only consider models like Continuous-Time LTI Differential Equation Model and Discrete-Time LTI Difference Equation Model. If a problem requires us to consider disturbances, we may fold them into the input, or else approximate them away.

As an example of how disturbance affects a model, consider the following model:

$$x_d[i+1] = 1.1x_d[i] + w_d[i] \tag{64}$$
$$x_d[0] = 1. \tag{65}$$

If we were to see how $x_d[i]$ evolves over time, we might see a plot like in fig. 2.

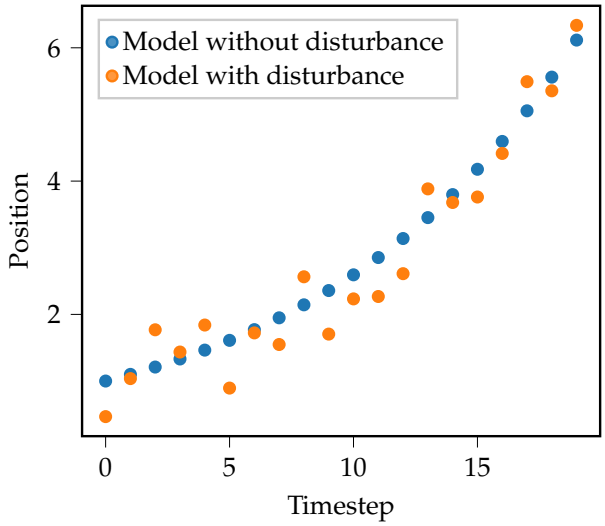

Figure 2: Evolution of a Discrete-Time LTI Difference Equation Model with and without disturbance

4 System Identification

Throughout this section, we will assume we are in Discrete-Time LTI Difference Equation Model, though all the concepts directly generalize to Continuous-Time LTI Differential Equation Model. A good test of how well you understand this section is to develop a system identification process for Continuous-Time LTI Differential Equation Model.

Thus far, we have learned how to convert an idealized Continuous-Time LTI Differential Equation Model we would obtain from theoretical knowledge of the system, into a Discrete-Time LTI Difference Equation Model we would be able to use in practice and implement on our computers.

Unfortunately, in many cases we would not even have a Continuous-Time LTI Differential Equation Model to work with, at least one where we would know the model parameters A_c and B_c with a large degree of certainty. As a result, we would not be able to discretize and get a computational model that we could use – because the discretization coefficients A_d and B_d would be unknown to us.

This is when system identification comes in. Given the assumption that we are using a Discrete-Time LTI Difference Equation Model, but no knowledge[3] of the coefficients A_d and B_d, system identification allows us to learn *estimates* $\widehat{A}_d$ and $\widehat{B}_d$ for our model parameters *from data*, say some data of the form $(\vec{x}_d[i], \vec{u}_d[i], \vec{x}_d[i+1])$ for some integer timesteps $i \in \{i_1, \ldots, i_\ell\}$.

4.1 Quantifying Error

To get a notion of the "best" estimate for A_d and B_d, we need to somehow define a notion of how bad our estimate $(\widehat{A}_d, \widehat{B}_d)$ is, so we can minimize it. This requires us to generalize the definition of the distance between vectors to matrices, and we do this by defining a new norm.

[3]System identification techniques can often be adapted to work with partial knowledge, but that is out of scope of this course.

Definition 16 (Frobenius Norm)
Let $A \in \mathbb{R}^{m \times n}$ be a matrix. The *Frobenius norm* of A is

$$\|A\|_F := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |A_{ij}|^2} \tag{66}$$

Note that

$$\|A\|_F = \|\texttt{unroll}(A)\| \tag{67}$$

where the right-hand side of the equation denotes taking the matrix A and unrolling it into one long column of length mn, and then taking the norm of this column. It is thus a "reasonable" idea for a matrix norm, as it denotes one notion of size of a matrix.

Concept Check: Show that $\|\cdot\|_F$ fulfills all the properties of a norm. You will probably need to check 16A notes again to remember what these properties are – the goal is just to get practice with the Frobenius norm.

We will use one important property of the Frobenius norm.

Proposition 17 (Frobenius Norm in terms of Column Norms)
Let $A \in \mathbb{R}^{m \times n}$ have columns $\vec{a}_1, \ldots, \vec{a}_n$. Then

$$\|A\|_F^2 = \sum_{j=1}^{n} \|\vec{a}_j\|^2. \tag{68}$$

Proof.

$$\|A\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} |A_{ij}|^2 \tag{69}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{n} |(\vec{a}_j)_i|^2 \tag{70}$$

$$= \sum_{j=1}^{n} \sum_{i=1}^{m} |(\vec{a}_j)_i|^2 \tag{71}$$

$$= \sum_{j=1}^{n} \|\vec{a}_j\|^2. \tag{72}$$

□

With that, we can quantify the error ℓ of our estimates with respect to the true parameters in terms of the Frobenius norm of the differences:

$$\text{error}((A_d, B_d), (\widehat{A}_d, \widehat{B}_d)) := \left\|A_d - \widehat{A}_d\right\|_F^2 + \left\|B_d - \widehat{B}_d\right\|_F^2 \tag{73}$$

$$= \left\| \begin{bmatrix} A_d \\ B_d \end{bmatrix} - \begin{bmatrix} \widehat{A}_d \\ \widehat{B}_d \end{bmatrix} \right\|_F^2, \tag{74}$$

is exactly the quantity we want to minimize, given data in the form of input/output pairs (but crucially no direct knowledge of A_d and B_d).

4.2 Least Squares for Error Minimization

We already have a tool to minimize vector norms. This result was stated in 16A, but is so central for our analysis that we restate it here.

Theorem 18 (Least Squares for Vector Estimation)
Suppose $D \in \mathbb{R}^{a\times b}$ has full column rank, and $\vec{s} \in \mathbb{R}^a$. If we define

$$\widehat{p} := (D^\top D)^{-1} D^\top \vec{s} \tag{75}$$

then[a]

$$\widehat{p} \in \operatorname*{argmin}_{\vec{p}\in\mathbb{R}^b} \|D\vec{p} - \vec{s}\|^2. \tag{76}$$

[a] The notation "$\in$ argmin" says that $\widehat{p}$ is one of the minimizers of the squared norm; it does not have to be the only one.

But, we are trying to estimate matrices, not vectors. This result does not quite apply in this case, so we will have to do some legwork to get it to apply.

Theorem 19 (Least Squares for Matrix Estimation)
Suppose $D \in \mathbb{R}^{a\times b}$ has full column rank, and $S \in \mathbb{R}^{a\times c}$. If we define

$$\widehat{P} := (D^\top D)^{-1} D^\top S \tag{77}$$

then

$$\widehat{P} \in \operatorname*{argmin}_{P\in\mathbb{R}^{b\times c}} \|DP - S\|_F^2. \tag{78}$$

Proof. Let $P \in \mathbb{R}^{b\times c}$ have columns $\vec{p}_1, \ldots, \vec{p}_c$, $S \in \mathbb{R}^{a\times c}$ have columns $\vec{s}_1, \ldots, \vec{s}_c$, and use proposition 17 to write

$$\begin{aligned}
\|DP - S\|_F^2 &= \left\| D\begin{bmatrix} \vec{p}_1 & \cdots & \vec{p}_c \end{bmatrix} - \begin{bmatrix} \vec{s}_1 & \cdots & \vec{s}_c \end{bmatrix} \right\|_F^2 && (79)\\
&= \left\| \begin{bmatrix} D\vec{p}_1 & \cdots & D\vec{p}_c \end{bmatrix} - \begin{bmatrix} \vec{s}_1 & \cdots & \vec{s}_c \end{bmatrix} \right\|_F^2 && (80)\\
&= \left\| \begin{bmatrix} D\vec{p}_1 - \vec{s}_1 & \cdots & D\vec{p}_c - \vec{s}_c \end{bmatrix} \right\|_F^2 && (81)\\
&= \sum_{i=1}^{c} \|D\vec{p}_i - \vec{s}_i\|^2. && (82)
\end{aligned}$$

Each of the summands is independent and, by theorem 18 may be minimized by using

$$\widehat{\vec{p}}_i := (D^\top D)^{-1} D^\top \vec{s}_i. \tag{83}$$

Then

$$\begin{aligned}
\widehat{P} &:= \begin{bmatrix} \widehat{p}_1 & \cdots & \widehat{p}_c \end{bmatrix} && (84)\\
&= \begin{bmatrix} (D^\top D)^{-1} D^\top \vec{s}_1 & \cdots & (D^\top D)^{-1} D^\top \vec{s}_c \end{bmatrix} && (85)\\
&= (D^\top D)^{-1} D^\top \begin{bmatrix} \vec{s}_1 & \cdots & \vec{s}_c \end{bmatrix} && (86)
\end{aligned}$$

$$= (D^\top D)^{-1} D^\top S \tag{87}$$

as desired. □

4.3 Least Squares for System Identification

In order to use the least squares theorems we proved, we need to convert our data into a data matrix D and an output matrix S, and have a parameter matrix P such that $DP \approx S$. Let us start with the simplest case: we have one data point $(\vec{x}_d[i], \vec{u}_d[i], \vec{x}_d[i+1])$. We already know that they are linearly related by

$$\vec{x}_d[i+1] = A_d \vec{x}_d[i] + B_d \vec{u}_d[i]. \tag{88}$$

Writing in matrix-vector form, we have

$$\vec{x}_d[i+1] = \begin{bmatrix} A_d & B_d \end{bmatrix} \begin{bmatrix} \vec{x}_d[i] \\ \vec{u}_d[i] \end{bmatrix}. \tag{89}$$

The issue with this is that the parameters A_d and B_d are on the *left* side of the matrix product, when we want them to be on the *right* side. To do this, we note that transposing reverses the order of factors in the product, or in mathematical terms $(AB)^\top = B^\top A^\top$, so we can push P to be on the right by taking transposes on both sides:

$$\vec{x}_d[i+1]^\top = \left(\begin{bmatrix} A_d & B_d \end{bmatrix} \begin{bmatrix} \vec{x}_d[i] \\ \vec{u}_d[i] \end{bmatrix} \right)^\top = \begin{bmatrix} \vec{x}_d[i]^\top & \vec{u}_d[i]^\top \end{bmatrix} \begin{bmatrix} A_d^\top \\ B_d^\top \end{bmatrix}. \tag{90}$$

Now we have a matrix of parameters on the right side of the equation, so we may define

$$P := \begin{bmatrix} A_d^\top \\ B_d^\top \end{bmatrix}. \tag{91}$$

If we have several timesteps $i_1, \ldots, i_\ell$, we can stack them to form the D and S matrices:

$$D := \begin{bmatrix} \vec{x}_d[i_1]^\top & \vec{u}_d[i_1]^\top \\ \vdots & \vdots \\ \vec{x}_d[i_\ell]^\top & \vec{u}_d[i_\ell]^\top \end{bmatrix} \in \mathbb{R}^{\ell \times (m+n)} \qquad S := \begin{bmatrix} \vec{x}_d[i_1+1]^\top \\ \vdots \\ \vec{x}_d[i_\ell+1]^\top \end{bmatrix} \in \mathbb{R}^{\ell \times n}. \tag{92}$$

Then we have

$$DP \approx S, \tag{93}$$

like we wanted.

The last questions to ask are:

- When does this D matrix has full column rank, i.e., when is $D^\top D$ invertible? The answer is that for D to be invertible we require $\text{rank}(D) = n + m$, i.e., the number of columns of D. So, we can only use least squares when $\ell \geq n + m$, because otherwise $\text{rank}(D) \leq \ell < n + m$. But even if $\ell \geq n + m$, we may have some rank deficiency, i.e., many samples may somehow be the same. For now it suffices to note that if we choose inputs with some randomness or use real data which has random noise, then D will be full column rank with high probability.[4]

[4] Interested readers who have taken courses in probability will note that, if the inputs are perturbed by a, say, multivariate Gaussian random variable, then D is full column rank with probability 1. This footnote is entirely out of scope of the course and is optional, but provides context as to what guarantees a reasonable error model can get.

- What should we do when $D^\top D$ is not invertible? In the case where there is at least one $\widehat{P}$ such that

$$D\widehat{P} = S \tag{94}$$

exactly (as opposed to approximately equal), then we are able to solve for each entry of $\widehat{P}$ via Gaussian elimination (say treating each column as a separate problem $D\vec{\widehat{p}}_i = \vec{s}_i$ and solving for $\vec{\widehat{p}}_i$, then constructing $\widehat{P}$). If not, we will have to resort to more advanced techniques, such as the *Moore-Penrose pseudoinverse* that is introduced later in the class.

This discussion motivates the following summary:

Method 20 (System Identification for Discrete-Time LTI Difference Equation Model)

Input: Integers n m, ℓ where $\ell \geq n + m$.
Input: Data $(\vec{x}_d[i], \vec{u}_d[i], \vec{x}_d[i+1])$ for $i \in \{i_1, \ldots, i_\ell\}$.
Output: Least-squares estimates $\widehat{A}_d$ and $\widehat{B}_d$.

let $D := \begin{bmatrix} \vec{x}_d[i_1]^\top & \vec{u}_d[i_1]^\top \\ \vdots & \vdots \\ \vec{x}_d[i_\ell]^\top & \vec{u}_d[i_\ell]^\top \end{bmatrix} \in \mathbb{R}^{\ell \times (n+m)}$
let $S := \begin{bmatrix} \vec{x}_d[i_1+1]^\top \\ \vdots \\ \vec{x}_d[i_\ell+1]^\top \end{bmatrix} \in \mathbb{R}^{\ell \times n}$
if $\text{rank}(D) < n + m$ **then**
    try Gaussian elimination to solve $DP = S$ column-by-column for a solution $\widehat{P} \in \mathbb{R}^{(n+m)\times n}$
    if any column Gaussian elimination fails **then**
        give up until we learn the Moore-Penrose pseudoinverse
    end if
else
    compute $\widehat{P} := (D^\top D)^{-1} D^\top S \in \mathbb{R}^{(n+m)\times n}$
end if
extract $\begin{bmatrix} \widehat{A}_d^\top \\ \widehat{B}_d^\top \end{bmatrix} := \widehat{P}$
return $(\widehat{A}_d, \widehat{B}_d)$

Concept Check: Write an equivalent algorithm for Continuous-Time LTI Differential Equation Model.

4.4 Example

One example is when we are trying to learn the model

$$x[t+1] = \frac{1}{3}x[t] + 5u[t]. \tag{95}$$

In this case, $A_d = \frac{1}{3}$ and $B_d = 5$. Let's suppose that we have the following data pairs, in the format $(x_d[i], u_d[i], x_d[i+1])$, taken from different runs of the same system:

$$\left(\frac{3}{5}, \frac{1}{5}, \frac{6}{5}\right) \tag{96}$$

$$\left(1, \frac{1}{3}, 2\right) \tag{97}$$

$$\left(\frac{1}{2}, \frac{3}{4}, \frac{47}{12}\right). \tag{98}$$

Then we would form the matrices

$$D := \begin{bmatrix} \frac{3}{5} & \frac{1}{5} \\ 1 & \frac{1}{3} \\ \frac{1}{2} & \frac{3}{4} \end{bmatrix} \qquad S := \begin{bmatrix} \frac{6}{5} \\ 2 \\ \frac{47}{12} \end{bmatrix}. \tag{99}$$

We would compute

$$D^\top D = \begin{bmatrix} \frac{161}{100} & \frac{497}{600} \\ \frac{497}{600} & \frac{2569}{3600} \end{bmatrix} \tag{100}$$

(if you're following along, please do this with a computer). We would see that $D^\top D$ is invertible and so we can use least squares. We would then get

$$(D^\top D)^{-1} D^\top = \begin{bmatrix} \frac{135}{238} & \frac{225}{238} & -\frac{4}{7} \\ -\frac{45}{119} & -\frac{75}{119} & \frac{12}{7} \end{bmatrix}. \tag{101}$$

Finally multiplying by S we would get

$$\widehat{P} := (D^\top D)^{-1} D^\top S = \begin{bmatrix} \frac{1}{3} \\ 5 \end{bmatrix} \tag{102}$$

which is exactly the true parameters P. We didn't have noise in our system, so $\widehat{P} = P$ exactly; otherwise, we would have some error.

5 Validation

We have built up a few techniques for creating new models in this note, but we are still missing one key component – ensuring that the model we created actually works.

The first way we can validate that our model actually works is by using it for what we want to do, and seeing whether it works. For models that are to be used for control, the goal is typically the successful control of the system. If we can successfully control our system by using our model, then the discretization or system identification is good enough for our purposes. This is the equivalent of testing a software system by deploying it in a production environment and verifying that it works.

However, in many learning contexts, we do not have access to the complete system at the time of building the model. If, for example, identifying the model via system identification is only one part of the larger engineering project, then the final outcome may not be related to the success or failure of the system identification (for instance, the hardware components may be non-functional).

To isolate the performance of the model-building component (similar to the so-called "unit testing" in software engineering), the key is to *test the prediction performance of the model using some data we did not use to learn the model in the first place* – the so-called "*validation dataset*", as we use it to validate the performance of our model, as opposed to the "*training dataset*" of data we used to learn the model.

Definition 21 (Training and Validation Data)

- *Training data* for a given model is the set of data samples used to construct estimates for the parameters.
- *Validation data* for a given model is the set of data samples that are not used in the estimation process, but are instead used to quantify the accuracy of the parameter estimates.

Normally we evaluate our model quantitatively using a so-called "*loss function*".

Definition 22 (Loss Function)
A loss function L takes in a model's input, output, and parameters, and quantifies the accuracy of the estimated output given the input, compared to the true output.

A very typical example is the squared norm loss for Discrete-Time LTI Difference Equation Model.

$$L(\vec{x}_d[i], \vec{u}_d[i], \vec{x}_d[i+1]; \widehat{A}_d, \widehat{B}_d) := \left\| \vec{x}_d[i+1] - (\widehat{A}_d \vec{x}_d[i] + \widehat{B}_d \vec{u}_d[i]) \right\|^2 \tag{103}$$

The end goal is to pick parameters *using the training dataset* that minimize the average "loss" across all data points *in the validation dataset*. For example, in the above case, we want to solve the problem

$$(\widehat{A}, \widehat{B}) := \underset{\widetilde{A}_d, \widetilde{B}_d}{\operatorname{argmin}} \frac{1}{\ell_{\text{val}}} \sum_{k=1}^{\ell_{\text{val}}} L(\vec{x}_d[i_{\text{val};k}], \vec{u}_d[i_{\text{val};k}], \vec{x}_d[i_{\text{val};k}+1]; \widetilde{A}_d, \widetilde{B}_d) \tag{104}$$

but our $\widehat{A}, \widehat{B}$ can only be functions of the training dataset.

These concepts will be discussed later in the course in greater detail when we study binary classification problems.

One final additional complexity you should keep in mind when using model-building techniques in the real world is that the model itself may change over time. To combat this additional level of difficulty, data is normally taken continuously to update the model via more sophisticated system identification techniques.

Contributors:
- Neelesh Ramachandran.
- Rahul Arya.
- Druv Pai.
- Anish Muthali.
- Anant Sahai.