EECS 16B Designing Information Devices and Systems II
Fall 2021

Note 9: Intro to Control and System ID

1 Overview

Recall the big-picture motivations for the course all the way back in Note 0. We want to understand what it takes to create artificial systems that interact with the real dynamical world. One important component of this is being able to act in the real world to achieve our goals — without the ability to act in the world, it is hard to have an impact. Interacting with the real world to achieve our objectives is the subject of an area called sometimes called "control." Control is one of the foundational disciplines for the areas of robotics, artificial intelligence, and machine learning, while also playing very critical roles in networking, high performance computer systems, as well as power systems and a host of other areas within EECS and beyond.

Our entry into control will be through what is sometimes called "model-based control" — where we leverage explicit models for the world in order to plan and execute our interventions. How do we model the dynamic world? So far, we have seen the language of differential equations as a way to do this. We've done our examples for circuits — since they are the simplest aspect of the physical world to model in terms of dynamics — but the language of differential equations is also used to model mechanical systems. However, we face a significant challenge when it comes to actually making systems: the real world we want to impact exists in continuous time but the computational platforms that we need to use (i.e. digital computers and microcontrollers) cannot act infinitely fast and so, in order to be reliable while being flexible enough to be programmable, must act at discrete intervals.

Given a linear differential equation model for a system, we know how we can discretize it and obtain a discrete-time system model. You've already seen this in discussion and the homework, and we'll recap the story briefly in this note as well. However, this relies on knowing physical parameters to infinite precision and many times, it is not worth investing that much effort in trying to model the system exactly by hand. *System Identification* is about using data collected about the inputs $\vec{u}$ and states $\vec{x}$ to attempt to determine the parameters of the model. This process is of great importance, because when we construct real-world electromechanical systems, it is practically impossible to determine all their behaviors theoretically - we must do so empirically. In practice, data is also taken continuously to update our models, since in the real world, the model itself can change over time. The fact that all learning from data necessarily involves some residual inaccuracy is also going to be important in how we think about the use of such models.

Modern system design leverages the ability to learn from data. Fortunately, EECS16A has already given you all the basic tools that you need to do this intelligently, as we see in this note.

2 From continuous-time models to discrete-time ones

We have seen how to model a continuous-time system:

$$\frac{d}{dt}\vec{x}_c(t) = A_c\vec{x}_c(t) + B_c\vec{u}_c(t). \tag{1}$$

What would be the counterpart for discrete-time? We could choose a time interval Δ, and decide to apply piecewise constant inputs for durations of length $\Delta > 0$ seconds, and then sample the state every Δ seconds to get a new discrete time system:

$$\vec{x}_d[i+1] = A_d\vec{x}_d[i] + B_d\vec{u}_d[i], \tag{2}$$

where $\vec{x}_d[i]$ is the value of $\vec{x}_c(i\Delta)$ and $\vec{u}_d[i]$ is the value of $\vec{u}_c(t)$ for the whole time interval $t \in [i, (i+1)\Delta)$.

Why do we need to do this? Because computers can only take in data in a sequence of snapshots taken with some spacing Δ between them (think of a movie at 24 frames per second) and for the same reason, can only change what computers put out into the world with some spacing Δ between outputs.

In discussion and the homework, we talked about discretization of a scalar differential equation. We originally did this so that we could take limits, but it is useful in its own right given that computers deal with the world in discrete steps. How does this work for systems of differential equations expressed in vector/matrix form? What is the relationship between A_d and B_d in eq. (2) and the A_c and B_c in eq. (1)?

It is easiest to see the exact relationship when there exists a coordinate system in which A_c is diagonal — $A_c = V\Lambda V^{-1}$. In the diagonal case, $\frac{d}{dt}\vec{\widetilde{x}}_c(t) = \Lambda\vec{\widetilde{x}}_c(t) + \widetilde{B}_c\vec{u}_c(t)$, the underlying system is basically a parallel set of scalar differential equations $\frac{d}{dt}\left(\vec{\widetilde{x}}_c(t)\right)_k = \lambda_k\left(\vec{\widetilde{x}}_c(t)\right)_k + \left(\widetilde{B}_c\vec{u}_c(t)\right)_k$. If $\lambda_k \neq 0$, this discretizes to $\left(\vec{\widetilde{x}}_d(i+1)\right)_k = e^{\Delta\lambda_k}\left(\widetilde{x}_d[i]\right)_k + \frac{e^{\Delta\lambda_k}-1}{\lambda_k}\left(\widetilde{B}_c\vec{u}_d[i]\right)_k$. If $\lambda_j = 0$, this discretizes to $\left(\vec{\widetilde{x}}_d[i+1]\right)_k = \left(\widetilde{x}_d[i]\right)_k + \left(\Delta\widetilde{B}_c\vec{u}_d[i]\right)_k$. Given that we have discretized each of the scalar differential equations in the transformed $\widetilde{x}$ coordinates, can you see how to derive the A_d and B_d by returning to the original coordinates?

For the remainder of this module, we will almost entirely disregard the fact that our system may in fact really be continuous, focusing entirely on questions related to our discrete-time model. After all, in reality, most electronic control systems (like the MSP that we will use to develop a "robot car") have some intrinsic Δ in their ability to sample $\vec{x}_c(t)$ and vary their output $\vec{u}(t)$, so even if we knew that $\vec{x}_c(t)$ varied within the interval Δ, we would not be able to measure or react to any variation within the time interval.

3 System Identification

Now that we are going to live in discrete time, we can drop the subscripts on x, A, B.

Let our linear system be:

$$\vec{x}[t+1] = A\vec{x}[t] + B\vec{u}[t] \tag{3}$$

where we observe each of the $\vec{x}[t]$. That is, t refers to the current timestep. From here on, it is understood that $\vec{x}[t]$ is discrete, so we will use subscripts to denote indices of the discrete-time vector (such as, $\vec{x}_1[t]$ is the value of the first component of $\vec{x}[t]$ at time t). Basically, there are going to be two different ways of indexing into these sequences of vectors. The $[t]$ will be used to index into a position in the sequence and the subscripts will be used to index into the vectors themselves.

Let's express our unknown matrices A and B in terms of scalars a_{ij}, b_{ij}, as follows:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{bmatrix} \tag{4}$$

Substituting into our relation for $\vec{x}[t]$, we break down our state, observation, and input vectors into their scalar components as well:

$$\begin{bmatrix} x_1[t+1] \\ x_2[t+1] \\ \\ x_n[t+1] \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1[t] \\ x_2[t] \\ \vdots \\ x_n[t] \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{bmatrix} \begin{bmatrix} u_1[t] \\ u_2[t] \\ \vdots \\ u_k[t] \end{bmatrix} \tag{5}$$

Now comes the interesting part. Recall that our unknowns are in fact all the a_{ij} and b_{ij} scalars, and our knowns are all the components of $\vec{x}$ and $\vec{u}$ (x_1, x_2, u_1, u_2, etc.) When solving linear systems, we know that we should put our unknowns into a vector. By considering each row of the above matrix separately (using r to index the row), we obtain scalar equations of the form:

$$x_r[t+1] = \begin{bmatrix} a_{r1} & a_{r2} & \dots & a_{rn} \end{bmatrix} \begin{bmatrix} x_1[t] \\ x_2[t] \\ \vdots \\ x_n[t] \end{bmatrix} + \begin{bmatrix} b_{r1} & b_{r2} & \dots & b_{rk} \end{bmatrix} \begin{bmatrix} u_1[t] \\ u_2[t] \\ \vdots \\ u_k[t] \end{bmatrix} \tag{6}$$

for all $1 \leq r \leq n$. Notice that the two terms on the right of the equation are really just inner products, producing a scalar. As the inner product is symmetric for real vectors ($\langle x, y \rangle = \langle y, x \rangle$), we can swap the rows and columns and rewrite the above equation as:

$$x_r[t+1] = \begin{bmatrix} x_1[t] & x_2[t] & \cdots & x_n[t] \end{bmatrix} \begin{bmatrix} a_{r1} \\ a_{r2} \\ \vdots \\ a_{rn} \end{bmatrix} + \begin{bmatrix} u_1[t] & u_2[t] & \cdots & u_k[t] \end{bmatrix} \begin{bmatrix} b_{r1} \\ b_{r2} \\ \vdots \\ b_{rk} \end{bmatrix} \tag{7}$$

again for all $1 \leq r \leq n$.

Combining the two terms on the right, we finally obtain the equation

$$x_r[t+1] = \begin{bmatrix} x_1[t] & x_2[t] & \cdots & x_n[t] & u_1[t] & u_2[t] & \cdots & u_k[t] \end{bmatrix} \begin{bmatrix} a_{r1} \\ a_{r2} \\ \vdots \\ a_{rn} \\ b_{r1} \\ b_{r2} \\ \vdots \\ b_{rk} \end{bmatrix} \tag{8}$$

again for all $1 \leq r \leq n$.

Notice that we have managed to place all the unknowns involving the rth element of the state at time t in a single vector. Since we now have a linear equation with our unknowns in a vector, are we done, since we can do Gaussian elimination to solve for the unknowns?

Unfortunately not. Notice that there are $n + k$ unknowns, but only a single equation. We could try to get

more equations by considering all values of r simultaneously (all the rows), but that would also bring in new unknowns, since we'd have to consider the a_{rj} and b_{rj} for *all* r. So is all hope lost?

No! Recall that our equation holds for *all* values of t, since we are assuming that our system's transition equation does not vary over time! Notice that for all timesteps $0 < t \leq T$, where T is the current time, the vector of unknowns remains the same. Thus, by considering all these values of t and stacking them vertically, we obtain the system

$$\begin{bmatrix} x_r[1] \\ x_r[2] \\ \vdots \\ x_r[T] \end{bmatrix} = \begin{bmatrix} x_1[0] & \cdots & x_n[0] & u_1[0] & \cdots & u_k[0] \\ x_1[1] & \cdots & x_n[1] & u_1[1] & \cdots & u_k[1] \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1[T-1] & \cdots & x_n[T-1] & u_1[T-1] & \cdots & u_k[T-1] \end{bmatrix} \begin{bmatrix} a_{r1} \\ a_{r2} \\ \vdots \\ a_{rn} \\ b_{r1} \\ b_{r2} \\ \vdots \\ b_{rk} \end{bmatrix} \tag{9}$$

which consists of T equations, not just 1! For $T \geq n + k$, we can use least squares to solve for our unknowns, and repeat the process for each value of r in order to fully determine our system. Typically, we create a matrix P of our unknowns across all values of r by stacking them horizontally to obtain

$$P = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nn} \\ b_{11} & b_{21} & \dots & b_{n1} \\ b_{12} & b_{22} & \dots & b_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1k} & b_{2k} & \dots & b_{nk} \end{bmatrix} = \begin{bmatrix} A^\top \\ B^\top \end{bmatrix}. \tag{10}$$

[← stacked horizontally]

We can follow a similar stacking procedure to obtain

$$D = \begin{bmatrix} x_1[0] & \cdots & x_n[0] & u_1[0] & \cdots & u_k[0] \\ x_1[1] & \cdots & x_n[1] & u_1[1] & \cdots & u_k[1] \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_1[T-1] & \cdots & x_n[T-1] & u_1[T-1] & \cdots & u_k[T-1] \end{bmatrix} \tag{11}$$

and

$$S = \begin{bmatrix} x_1[1] & x_2[1] & \cdots & x_n[1] \\ x_1[2] & x_2[2] & \cdots & x_n[2] \\ \vdots & \ddots & \ddots & \vdots \\ x_1[T] & x_2[T] & \cdots & x_n[T] \end{bmatrix} \tag{12}$$

Thus, we can combine all our values of r into the single approximate equation

$$DP \approx S$$

and solve this using least squares to obtain

$$P = \left(D^\top D\right)^{-1} D^\top S \tag{13}$$

Why do we use least-squares here? Because we are going to assume that any measurement errors or unmodeled terms are all relatively small. So it makes sense to pick a solution that has a small residual, and least squares gives that to us. (We can see in another note how we could make such solutions robust to a few points being hit with something much larger.)

Recall that $D^\top D$ is invertible exactly when D has linearly independent columns. We will discuss what happens when the columns made out of the x_r are dependent later — for now, we will simply comment that choosing our inputs randomly will ensure with high probability that the input columns are all linearly independent both of each other and of the earlier x_r columns.

The above recipe is pretty generic. We write out our model and figure out what are parameters that we need to learn. Then, we use the data that we have collected to set up systems of approximate equations in the unknown parameters that we need to learn. Finally, we use least-squares to solve for those parameters. If our measurements are of high quality and our model is a reasonable match to reality, then we will get estimated values for the parameters that work well.

Contributors:

- Neelesh Ramachandran.
- Rahul Arya.
- Anant Sahai.

