EECS 16B Designing Information Devices and Systems II
Fall 2019

Note: Linearization

Overview

So far, we have spent a great deal of time studying linear systems described by differential equations of the form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) = A\vec{x}(t) + B\vec{u}(t) + \vec{w}(t)$$

or linear difference equations (aka linear recurrence relations) of the form:

$$\vec{x}(t+1) = A\vec{x}(t) + B\vec{u}(t) + \vec{w}(t).$$

Here, the linear difference equations came from either discretizing a continuous model or were learned from data.

In the real world, however, it is very rare for physical systems to perfectly obey such simple linear differential equations. The real world is generally continuous, but our systems will obey nonlinear differential equations of the form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x} = \vec{f}(\vec{x}(t), \vec{u}(t)) + \vec{w}(t),$$

where $\vec{f}$ is some non-linear function. Through linearization, we will see how to approximate general systems as locally linear systems, which will in turn allow us to apply all the techniques we have developed so far.

1 Taylor Expansion

First, we must recall how to approximate the simplest nonlinear functions — scalar functions of one variable — as linear functions. To do so, we will use the Taylor expansion we learned in calculus. This expansion tells us that for well-behaved functions $f(x)$, we can write

$$f(x) = f(x_0) + \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=x_0}(x-x_0) + \frac{1}{2}\left.\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right|_{x=x_0}(x-x_0)^2 + \cdots$$

near any particular point $x = x_0$, known as the *expansion point*. By truncating this expansion after the first few terms, we can obtain a locally polynomial approximation for f of a desired degree d. For instance, we may take just the first two terms to linearly approximate $f(x)$ as

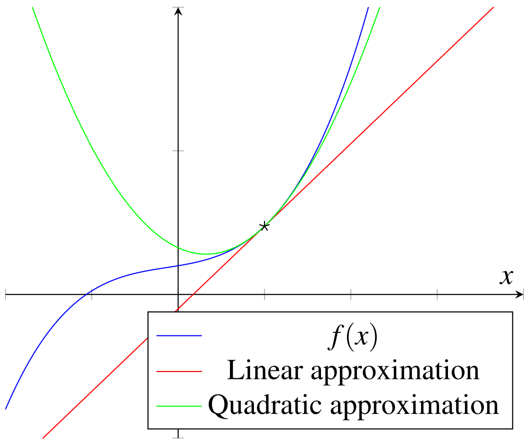
$$f(x) \approx f(x_0) + \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=x_0}(x-x_0).$$

It is further known that, when working in the "neighborhood" of $x = x_0$ (i.e. in a small interval around that point), the error of such a truncated approximation is bounded if the function is well behaved[1]. The exact

[1]For example, it has a bounded derivative beyond the level at which we truncate in the neighborhood of interest.

magnitude of this bound can be determined[2] but is not very important for our purposes — the important part is that it is indeed bounded.

Before going further, let's see what these approximations look like.



We have plotted a nonlinear function $f(x)$ and its linear approximation constructed around $x_0 = 0.5$. Notice that though the approximation deviates greatly from $f(x)$ over the whole domain, in the close neighborhood of x_0, the error of the approximation is very small. Including one more term of our Taylor expansion, we obtain the quadratic approximation. Qualitatively, this approximation matches our function better in the neighborhood of our expansion point. In practice, we rarely go beyond the quadratic terms when trying to find a polynomial approximation of a function in a local region.

The ability to approximate is generally good because it lets us attack a problem that we don't know to solve by approximating it using another problem that we do know how to solve. But this only makes engineering sense if we have some way of dealing with the impact of the approximation error. Control provides a very useful example for this kind of thinking. If we approximate the f in a nonlinear differential equation by a linear function, why can we trust that everything will work out when we use this approximation to control? The answer is provided by the disturbance term $w(t)$ — our feedback control design for the linearized system should be able to reject bounded disturbances. In the neighborhood of the expansion point, our approximation differs from the true function by a small bounded amount. This approximation error can therefore be absorbed into the existing disturbance term — we just make the disturbance a little bigger. As long as the feedback controlled system stays within the region that the approximation is valid, we are fine. In effect, we pass the buck and let the feedback control deal with the approximation error. We'll see in a later note on classification how iteration provides another way to deal with the impact of approximation errors.

2 Multivariate Scalar-Valued Functions

Unfortunately, we can't directly apply the Taylor expansions that we learned in our calculus courses to solve our problem of linearizing a nonlinear system. Why is that? Well, so far we only know how to linearize

[2]Remember from your calculus courses — this is an application of the mean value theorem. You can take the maximum absolute value for the (presumably bounded) next derivative and use it in a virtual "next term" of the expansion to get an explicit bound. The fact that the factorial $(k+1)!$ in the denominator is very rapidly growing is what makes this bound get very small as we add terms in the expansion. But for us, the more important fact is that the bound includes $(x-x_0)^{k+1}$. This is what makes the bound get very small in a tight enough neighborhood around x_0. This tells us that we can use appoximations that are not that high order (like linear) and still have a good local approximation. If the second derivative is bounded by 2, then in a neighborhood of distance 0.1 from x_0, the linear approximation will be good to a precision of 0.01.

scalar functions of one variable. But any nontrivial control system has at least *two* variables: the state and the control input! Even with just a scalar state and one-dimensional control input, our system would look like something of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = f(x(t), u(t)).$$

So we need to figure out how to deal with $f(x, u)$. Ideally, we'd be able to pick an expansion point (x_0, u_0), around which we could linearize f as

$$f(x_0 + \delta x, u_0 + \delta u) \approx f(x_0, u_0) + a \times \delta x + b \times \delta u,$$

where a and b depend only on the function f and the x_0 and u_0. It makes sense that a should somehow represent the "sensitivity" of f to variations in x, and b the sensitivity of f with respect to variations in u.

To get a better understanding of how we can compute these two quantities, it will help to consider an example. Let $f(x, y) = x^3u^2$. Then we see that, near an expansion point (x_0, u_0), we can rewrite

$$\begin{aligned} f(x_0 + \delta x, u_0 + \delta u) &= (x_0 + \delta x)^3(u_0 + \delta u)^2 \\ &= (x_0^3 + 3x_0^2\delta x + 3x_0\delta x^2 + \delta x^3)(u_0^2 + 2u_0\delta_u + \delta u^2). \end{aligned}$$

Imagine that we are very near this expansion point, so $|\delta x|, |\delta u| \ll x_0, u_0$. In other words, we can treat δx and δu as two comparably tiny quantities, that are both much smaller than x_0 or u_0. Then which terms in the above product (when expanded out fully) are most significant?

Well, any term involving δx or δu will immediately be made small, so the most significant term is $x_0^3u_0^2$. This term is just a constant, and isn't very interesting since it doesn't vary at all as we move around the neighborhood of our expansion point. So what can we get next? Well, we want as few δx and δu's as possible in our terms. We've already got the only term with zero of them, so the next most significant terms will be those involving exactly one tiny quantity: $3x_0^2u_0^2\delta x$ and $2u_0x_0^3\delta u$. Every other term will have products of tiny terms and we know that products of tiny numbers are even tinier numbers. If you will recall, this is the same type of reasoning that you saw in your calculus course when you first encountered derivatives.

Thus, using only these most significant terms, we obtain the approximation

$$x^3u^2 \approx x_0^3u_0^2 + 3x_0^2u_0^2\delta x + 2u_0x_0^3\delta u.$$

The $x_0^3u_0^2$ term clearly corresponds to the $f(x_0, u_0)$ term in our desired linearization. In a similar manner, $3x_0^2u_0^2$ corresponds to the coefficient a and $2u_0x_0^3$ to the coefficient b.

How could we have obtained these coefficients directly from f? The key insight is the following. Imagine that u was in fact a constant, so our function was just the scalar valued function of a single scalar variable $f(x) = x^3u^2$. Then what is its derivative? Well, since u^2 is being treated as a constant, its derivative is $3x^2u^2$, when when evaluated at our expansion point yields exactly the coefficient a. Similarly, by treating x as a constant and writing our nonlinear function as $f(u) = x^3u^2$, we can differentiate to obtain $2ux^3$, which when evaluated at the expansion point gives us b.

In other words, we obtain the desired coefficients by treating all the variables except the one of interest as a constant, and then differentiating f with respect to the remaining variable.

These quantities are known as the *partial derivatives* of f, and are denoted as

$$a = \left.\frac{\partial f}{\partial x}\right|_{x=x_0, u=u_0} \text{ and } b = \left.\frac{\partial f}{\partial u}\right|_{x=x_0, u=u_0}.$$

It turns out that the pattern we saw with our example generalizes, so we can linearize an arbitrary (well-behaved) scalar-valued function of multiple variables about an expansion point (x_0, u_0) as

$$f(x,u) = f(x_0,u_0) + \left(\frac{\partial f}{\partial x}\bigg|_{x=x_0,u=u_0} \right)(x - x_0) + \left(\frac{\partial f}{\partial u}\bigg|_{x=x_0,u=u_0} \right)(u - u_0),$$

with this expansion generalizing in the natural manner to functions with more arguments.

3 Vector-Valued Functions

With partial derivatives in hand, we can return to our original problem of linearizing a vector-valued function $\vec{f}(\vec{x},\vec{u})$.

First, we should ask ourselves what the desired form of the linearization could be. Recalling our experience with linear systems, we'd probably like our linearization near an expansion point $(\vec{x}_0,\vec{u}_0)$ to look something like

$$\vec{f}(\vec{x},\vec{u}) \approx \vec{f}(\vec{x}_0,\vec{u}_0) + A(\vec{x} - \vec{x}_0) + B(\vec{u} - \vec{u}_0),$$

where A and B are some matrices that presumably depend on $\vec{f}$ and $\vec{x}_0$ and $\vec{u}_0$.

Let our state $\vec{x}$ be n-dimensional and our control $\vec{u}$ be k-dimensional. Since $\frac{\mathrm{d}}{\mathrm{d}t}\vec{x} = \vec{f}(\vec{x},\vec{u})$, the output of $\vec{f}$ must also be n-dimensional.

Thus, by the rules of matrix multiplication, we know that A must be an $n \times n$ matrix and B an $n \times k$ matrix, for the dimensions to all work out.

But what are the contents of these two matrices? To determine this, we will look at our vectors elementwise, expressing the function we wish to linearize as

$$\vec{f}(\vec{x},\vec{u}) = \begin{bmatrix} f_1(\vec{x},\vec{u}) \\ f_2(\vec{x},\vec{u}) \\ \vdots \\ f_n(\vec{x},\vec{u}) \end{bmatrix},$$

where the f_i are each scalar-valued functions. Let's now consider a single one of these f_i, and try to linearize it about an expansion point. Note that though we write each f_i as functions of the vectors $\vec{x}$ and $\vec{u}$, they are really simply functions of the $n+k$ individual scalars $x[1],\ldots,x[n],u[1],\ldots,u[k]$ that form the components of the two vector-valued arguments. Thus, we can use the linearization techniques we saw earlier to approximate

$$f_i(\vec{x},\vec{u}) \approx f_i(\vec{x}_0,\vec{u}_0) + \sum_{j=1}^{n} \left(\frac{\partial f_i}{\partial x[j]}\bigg|_{(\vec{x}_0,\vec{u}_0)} \right)(x[j] - x_0[j]) + \sum_{j=1}^{k} \left(\frac{\partial f_i}{\partial u[j]}\bigg|_{(\vec{x}_0,\vec{u}_0)} \right)(u[j] - u_0[j]).$$

For notational convenience, for the remainder of this section we will stop explicitly writing down where we evaluate the partial derivatives, but they are always to be evaluated at the expansion point $(\vec{x}_0,\vec{u}_0)$. We can rearrange the above linearization as the matrix multiplication

$$f_i(\vec{x},\vec{u}) \approx f_i(\vec{x}_0,\vec{u}_0) + \begin{bmatrix} \frac{\partial f_i}{\partial x[1]} & \cdots & \frac{\partial f_i}{\partial x[n]} \end{bmatrix}(\vec{x} - \vec{x}_0) + \begin{bmatrix} \frac{\partial f_i}{\partial u[0]} & \cdots & \frac{\partial f_i}{\partial u[k]} \end{bmatrix}(\vec{u} - \vec{u}_0).$$

The two row vectors above can be thought of as the derivatives[3] of $f_i(\vec{x},\vec{u})$ with respect to $\vec{x}$ and $\vec{u}$. We can denote these rows by $\frac{\partial f_i}{\partial \vec{x}}$ and $\frac{\partial f_i}{\partial \vec{u}}$ if we would like.

Now, stacking the above linearization to obtain a linearization for the original vector-valued $\vec{f}(\vec{x},\vec{u})$, we obtain

$$\vec{f}(\vec{x},\vec{u}) \approx \vec{f}(\vec{x}_0,\vec{u}_0) + \begin{bmatrix} \frac{\partial f_1}{\partial x[1]} & \cdots & \frac{\partial f_1}{\partial x[n]} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x[1]} & \cdots & \frac{\partial f_n}{\partial x[n]} \end{bmatrix} (\vec{x}-\vec{x}_0) + \begin{bmatrix} \frac{\partial f_1}{\partial u[1]} & \cdots & \frac{\partial f_1}{\partial u[n]} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u[1]} & \cdots & \frac{\partial f_n}{\partial u[n]} \end{bmatrix} (\vec{u}-\vec{u}_0),$$

so we have solved for $A = \begin{bmatrix} \frac{\partial f_1}{\partial \vec{x}} \\ \vdots \\ \frac{\partial f_n}{\partial \vec{x}} \end{bmatrix}$ and $B = \begin{bmatrix} \frac{\partial f_1}{\partial \vec{u}} \\ \vdots \\ \frac{\partial f_n}{\partial \vec{u}} \end{bmatrix}$ in our linear original model.

Notice that it makes sense to think of A as the partial derivative of the vector-valued function $\vec{f}$ with respect to the vector input $\vec{x}$ and to just define $A = \frac{\partial \vec{f}}{\partial \vec{x}}$ and $B = \frac{\partial \vec{f}}{\partial \vec{u}}$. Once again it makes sense for these derivatives to be matrices since a matrix acting on a vector returns a vector.

One great development is that modern software frameworks like PyTorch and TensorFlow have automatic support for computing derivatives of functions that are expressed naturally using code. This means that all these ideas around linearization are now quite easy to implement — you don't have to compute all the derivatives by hand in practice.

4 DC Operating Points

Now, let's see how this linearization plugs back into our original differential equation (the discrete-time case behaves analogously for the purposes of linearization, but we will focus on the continuous-time case to avoid duplicating calculations). We see that

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) = \vec{f}(\vec{x}_0,\vec{u}_0) + \begin{bmatrix} \frac{\partial f_1}{\partial x[1]} & \cdots & \frac{\partial f_1}{\partial x[n]} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x[1]} & \cdots & \frac{\partial f_n}{\partial x[n]} \end{bmatrix} (\vec{x}(t)-\vec{x}_0) + \begin{bmatrix} \frac{\partial f_1}{\partial u[1]} & \cdots & \frac{\partial f_1}{\partial u[n]} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u[1]} & \cdots & \frac{\partial f_n}{\partial u[n]} \end{bmatrix} (\vec{u}(t)-\vec{u}_0) + \vec{w}(t).$$

Notice that even though we are using an approximation, we have used an equals sign $=$ above. This is because we have folded the approximation error into the disturbance term $\vec{w}(t)$.

So have we gotten this into a form that we know how to deal with?

The first apparent problem is that we have $\vec{x}(t)-\vec{x}_0$ on the right, but $\vec{x}(t)$ on the left. By rewriting $\vec{x}(t) =$

[3]In general, the derivative of a scalar valued function of a vector has to be a row vector. This is because it needs to act on an change in a column vector and return a scalar that represents the change to the function. Rows multiply columns to give scalars. For those of you who might have taken a course in multivariable calculus, it is vital that you not confuse the gradient ∇f with the derivative. The gradient is also a column vector and is the transpose of the derivative of the function f.

$(\vec{x}(t)-\vec{x}_0)+\vec{x}_0$ and expanding out the derivative, we obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}_0+\frac{\mathrm{d}}{\mathrm{d}t}(\vec{x}(t)-\vec{x}_0)=\vec{f}(\vec{x}_0,\vec{u}_0)+\begin{bmatrix}\frac{\partial f_1}{\partial x[1]} & \cdots & \frac{\partial f_1}{\partial x[n]}\\ \vdots & \ddots & \vdots\\ \frac{\partial f_n}{\partial x[1]} & \cdots & \frac{\partial f_n}{\partial x[n]}\end{bmatrix}(\vec{x}(t)-\vec{x}_0)+\begin{bmatrix}\frac{\partial f_1}{\partial u[1]} & \cdots & \frac{\partial f_1}{\partial u[n]}\\ \vdots & \ddots & \vdots\\ \frac{\partial f_n}{\partial u[1]} & \cdots & \frac{\partial f_n}{\partial u[n]}\end{bmatrix}(\vec{u}(t)-\vec{u}_0)+\vec{w}(t).$$

Essentially, we can treat $\vec{x}(t)-\vec{x}_0$ as our state vector and $\vec{u}(t)-\vec{u}_0$ as our control input. We're not yet, though. There are still terms on both sides of the equation that aren't present in the linear models we have previously looked at: $\frac{\mathrm{d}\vec{x}_0}{\mathrm{d}t}$ on the left, and $\vec{f}(\vec{x}_0,\vec{u}_0)$ on the right.

Notice that both these terms depend only on the choice of expansion point. When trying to linearize a system, it is our job to pick expansion points such that these two terms cancel out, leaving us with only the linear system in $\vec{x}(t)-\vec{x}_0$ and $\vec{u}(t)-\vec{u}_0$.

The simplest (and more common) case is that $\vec{x}_0$ is constant with time. In this case, its derivative is zero. Therefore, we need to choose our expansion point such that

$$\vec{f}(\vec{x}_0,\vec{u}_0)=\frac{\mathrm{d}\vec{x}_0}{\mathrm{d}t}=\vec{0}. \tag{1}$$

There are generally many potential solutions to the above nonlinear system of equations (1). Which one you choose depends on the context or design goal. For example, one might want to find a feedback control law that stabilizes a nonlinear system about a particular state $\vec{x}_0$. In that case, you will want to solve for the the corresponding $\vec{u}_0$ that satisfies (1) with that particular state $\vec{x}_0$. It can be thought of as the control that would keep the original nonlinear system held at that particular point in the absence of all disturbances. To find it, we can use numerical[4] or graphical[5] methods to solve the nonlinear system of equations (1).

Any particular $(\vec{x}_0,\vec{u}_0)$ pair that solves (1), is known as a *DC operating point*. These are also called *stationary points* or *equilibria*. When we linearize a dynamical system around such an equilibrium, the resulting linear system will have constant A,B matrices because $(\vec{x}_0,\vec{u}_0)$ will be constants. The system's stability can be checked by looking at the eigenvalues of A and the local controllability can be checked by looking at A,B together. At this point, if the system is controllable, we can use feedback control to place the eigenvalues of $A+BK$ where we want and thereby get a feedback control law that will stabilize the original nonlinear system about the chosen operating point. Here it is important to remember what everything means. The actual control to be applied will be $\vec{u}_0+K(\vec{x}(t)-\vec{x}_0)$ — because the K has been calculated for the linearized system and we need to apply a control in the original system.

The ability to linearize is what makes linear control practical since many (if not most) real-world systems

[4]So, how do you solve nonlinear systems of equations like (1)? This can be challenging at times but fortunately software packages exist for you to use. For example, SciPy has scipy.optimize.fsolve for you. In the homework, you will see a problem that shows you how iterative methods can be used to solve a general $\vec{g}(\vec{\theta})=\vec{0}$. These begin with a guess $\vec{\theta}_0$ for a solution, and then proceed iteratively. One intuitive way (that you will explore in the homework) is to linearize the function $\vec{g}$ around $\vec{\theta}$. This gives rise to a linear system of equations $\vec{g}(\vec{\theta}_0)+\vec{g}'(\vec{\theta}_0)(\vec{\theta}_0+\vec{\delta\theta})=\vec{0}$. Here, $\vec{g}'(\vec{\theta}_0)$ denotes thederivative of $\vec{g}$ with respect to $\vec{\theta}$, i.e. the matrix $\frac{\partial\vec{g}}{\partial\vec{\theta}}$. We can solve this linear system of equations using what you know for a candidate update $\vec{\delta\theta}$. At this point, we can update our guess and iterate. We could update our new guess $\vec{\theta}_{i+1}$ to be $\vec{\theta}_i+\vec{\delta\theta}$ itself. Or we could be more conservative (under the belief that the linearization was only valid in a small neighborhood) and just take a little step in that direction to $\vec{\theta}_i+\eta\vec{\delta\theta}$ where η is some small constant that is less than 1. In general, by iterating such an algorithm with a small enough step-size η, we find our way to a root. This is also the approach that we will take when we apply linearization ideas to the problem of classification in the next note.

[5]By graphical, we literally mean methods that involve drawing the graph of the function and seeing where it crosses zero or related intuitive approaches.

that we encounter are fundamentally nonlinear. The linearized system of differential equations can also be discretized to give rise to a discrete-time linear control system. Here, the critical choice is that we must choose a discretization time interval length that is small enough so that the approximation remains valid. The system must not move outside of the region of approximation validity (i.e. the neighborhood where the approximation error induced by linearization is within the desired bound) during the interval between taking samples. As long as we sample fast enough, we are free to design our control laws in discrete-time.

5 Linearization beyond DC operating points — towards trajectories

You've seen in the homework that we can do linear feedback control to keep a linear system stably following a desired chosen open-loop trajectory. Looking at the equations earlier, you should notice that we can just as well linearize in the neighborhood of a nominal trajectory $(\vec{x}_0(t), \vec{u}_0(t))$ that solves

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}_0(t) = \vec{f}(\vec{x}_0(t), \vec{u}_0(t)). \tag{2}$$

How do we find such nominal solutions? That is a bit out of the scope of this course but it turns out that the same kinds of iterative numerical approaches that work for finding DC operating points can be leveraged here as well. We linearize locally, solve, and then relinearize and resolve repeatedly.

However, there is one more challenge that occurs when we use linearization around a nominal trajectory — the resulting A and B matrices are now generically functions of time. So does that mean we are doomed? Not at all! While one has to be careful, often we can succeed by taking the same modeling philosophy that we have taken so far. If we assume that we are sampling the system fast enough, we can hope that the A and B matrices don't change too much from one step to the next. As a result, we can approximate the A matrices as being locally constant over many samples and just fold the approximation error into the disturbances. Here, we need to be a bit more careful because any error in A (due to time-variation) multiplies the deviation of the state and we need the product to stay small. If the state is small and the Frobenius norm of the change in A is small, the product will also be small. Similarly for B — but here we must ensure that the control that we apply times the change in B stays small. This means that the feedback controls should be gentle enough. You will learn how to navigate the resulting tension in more advanced control courses.

Finally, what about planning a nominal trajectory? Here, it turns out that the minimum-norm approach that you have learned can be iterated to get plans. The reason is that there was nothing in the approach that required the A and B matrices to stay constant. So it is possible to guess a set of controls to get to a destination, linearize, find a minimum norm solution, and then iterate this process to get a plan. Not all starting guesses will work, and so sometimes, more brute force guessing or smarter heuristics have to be used to speed this process up. This is also a topic for more advanced robotics courses.

Contributors:

- Rahul Arya.
- Anant Sahai.