EECS 16B Designing Information Devices and Systems II
Fall 2021

Note 19: Linearization and Quadratic Approximation

Overview

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

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) = A\vec{x}(t) + B\vec{u}(t) + \vec{w}(t) \tag{1}$$

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

$$\vec{x}[i+1] = A\vec{x}[i] + B\vec{u}[i] + \vec{w}[i]. \tag{2}$$

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 linear differential equations. For example:

(a) Up until now, we have considered transistors to be binary – turning off or on at some voltage differential. But in reality, transistors work continuously, and the relevant governing equations are highly nonlinear. For example, the Wikipedia article on BJTs has some of them.

(b) Robotics and control in general has highly nonlinear dynamics. The reason for this is because the system has a variety of geometric relationships that must hold, and so there are many trigonometric functions involved, which are all non-linear but have good linear approximations. An example of this analysis may be found when applying control to stabilize an inverted pendulum on a cart.

(c) Machine learning and optimization also have highly nonlinear systems. In particular, the concept of *gradient flow* sets up the differential equation $\frac{\mathrm{d}}{\mathrm{d}t}x(t) = -\frac{\mathrm{d}}{\mathrm{d}x(t)}f(x(t))$, and if one solves for $x(t)$ then one can characterize the whole process of optimization via *gradient descent*. In modern machine learning, such as in neural networks, the function f is nonlinear.

Don't worry, we don't expect you to know anything from these examples that hasn't been taught elsewhere in the course! They are just there for flavor.

Our systems will obey nonlinear differential equations of the form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x} = \vec{f}(\vec{x}(t), \vec{u}(t)) + \vec{w}(t), \tag{3}$$

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. We will then expand our theory to quadratic approximations as a straightforward generalization.

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 analytic functions $f(x)$ (functions that can be infinitely differentiated everywhere), we can write

$$f(x) = f(x^\star) + f'(x^\star)(x - x^\star) + \frac{1}{2}f''(x^\star)(x - x^\star)^2 + \dots \tag{4}$$

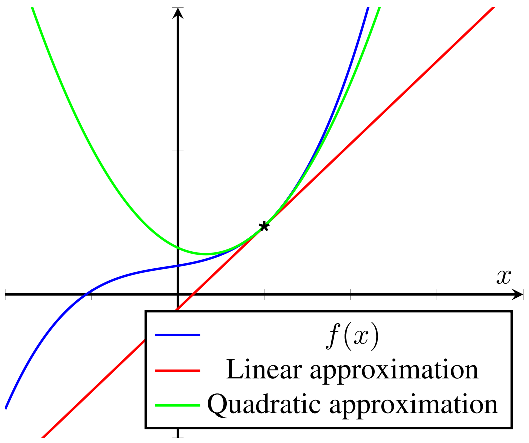
$$= f(x^\star) + \frac{\mathrm{d}f}{\mathrm{d}x}(x^\star) \cdot (x - x^\star) + \frac{1}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x^\star) \cdot (x - x^\star)^2 + \dots \tag{5}$$

near any particular point $x = x^\star$, known as the *expansion point*.[1] 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 $f(x)$ at $x^\star$ as

$$f(x) \approx f(x^\star) + \frac{\mathrm{d}f}{\mathrm{d}x}(x^\star) \cdot (x - x^\star). \tag{6}$$

It is further known that, when working in the "neighborhood" of $x = x^\star$ (i.e. in a small interval around that point), the error of such a truncated approximation is bounded if the function is well behaved[2]. The exact magnitude of this bound can be determined[3] 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 $x^\star = 0.5$. Notice that though the approximation deviates greatly from $f(x)$ over the whole domain, in the close neighborhood

[1]Note that the two expressions above are just two different notations for the same thing!

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

[3]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 the expansion to get an explicit upper 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 $(x - x^\star)^{k+1}$. This is what makes the bound get very small in a tight enough neighborhood around $x^\star$. This tells us that we can use approximations 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 $x^\star$, the linear approximation will be good to a precision of 0.01.

of $x^\star$, 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 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

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = f(x(t), u(t)). \tag{7}$$

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

$$f(x^\star + \delta x, u^\star + \delta u) \approx f(x^\star, u^\star) + a \cdot \delta x + b \cdot \delta u, \tag{8}$$

where a and b depend only on the function f and the $x^\star$ and $u^\star$. 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. [4]

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

$$\begin{aligned} f(x^\star + \delta x, u^\star + \delta u) &= (x^\star + \delta x)^3 (u^\star + \delta u)^2 & (9)\\ &= (x^{\star 3} + 3x^{\star 3} \cdot \delta x + 3x^\star \cdot (\delta x)^2 + (\delta x)^3)(u^{\star 2} + 2u^\star \cdot \delta u + (\delta u)^2) & (10)\\ &= x^{\star 3} u^{\star 2} + 3x^{\star 2} u^{\star 2} \cdot \delta x + 2x^{\star 3} u^\star \cdot \delta u + \cdots & (11) \end{aligned}$$

Why did we drop a lot of terms when making our approximation?

Imagine that we are very near this expansion point, so $|\delta x|, |\delta u| \ll x^\star, u^\star$. In other words, we can treat δx and δu as two comparably tiny quantities, that are both much smaller than $x^\star$ or $u^\star$. Then which terms in

[4]Here note that δx and δu are one variable each, not the product of two variables – in general, δz usually means "a small quantity that's of the same type as z". This notation comes from calculus, where we reason with ϵ and δ in the following way: "for every ϵ there exists a δ such that...". Usually δ is used in the argument of the function, in these contexts.

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 $x^{\star 3}u^{\star 2}$. 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: $3x^{\star 2}u^{\star 2} \cdot \delta x$ and $2u^{\star}x^{\star 3} \cdot \delta u$. 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

$$x^3u^2 \approx x^{\star 3}u^{\star 2} + 3x^{\star 2}u^{\star^2} \cdot \delta x + 2x^{\star 3}u^{\star} \cdot \delta u. \tag{12}$$

The $x^{\star 3}u^{\star 2}$ term corresponds to the $f(x^\star, u^\star)$ term in our desired linearization. In a similar manner, $3x^{\star 2}u^{\star 2}$ corresponds to the coefficient a and $2u^\star x^{\star 3}$ to the coefficient b.

How could we have obtained these coefficients directly from f?

2.1 Partial Derivatives

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 $3x^2u^2$. Similarly, by treating x as a constant and writing our nonlinear function as $f(u) = x^3u^2$, we can differentiate to obtain $2ux^3$. These quantities are known as the *partial derivatives* of f, and are denoted as $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial u}$ in analogy with the single-variable notation $\frac{\mathrm{d}f}{\mathrm{d}x}$, or alternatively the more compact notation f_x, f_u in analogy with the single-variable notation f'.

Now when we evaluate the partial derivatives at our expansion point $(x^\star, u^\star)$, we exactly get the terms from above. 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. Thus, we get

$$a = \frac{\partial f}{\partial x}(x^\star, u^\star) = f_x(x^\star, u^\star) \qquad \text{and} \qquad b = \frac{\partial f}{\partial u}(x^\star, u^\star) = f_u(x^\star, u^\star). \tag{13}$$

Then we can linearize any well-behaved scalar-valued function of 2 variables about an expansion point $(x^\star, u^\star)$ as

$$f(x,u) \approx f(x^\star, u^\star) + \frac{\partial f}{\partial x}(x^\star, u^\star) \cdot (x - x^\star) + \frac{\partial f}{\partial u}(x^\star, u^\star) \cdot (u - u^\star) \tag{14}$$

If we had more than 2 variables as the input, and instead a vector of variables, we can generalize this to get the linearization of $f(\vec{x})$ around an expansion point $\vec{x}^\star$:

$$f(\vec{x}) \approx f(\vec{x}^\star) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\vec{x}^*) \cdot (x_i - x_i^\star) \tag{15}$$

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 $(\vec{x}^{\star}, \vec{u}^{\star})$ to look something like

$$\vec{f}(\vec{x}, \vec{u}) \approx \vec{f}(\vec{x}^{\star}, \vec{u}^{\star}) + A(\vec{x} - \vec{x}^{\star}) + B(\vec{u} - \vec{u}^{\star}), \tag{16}$$

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

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

$$\vec{f}(\vec{x}, \vec{u}) = \begin{bmatrix} f_1(\vec{x}, \vec{u}) \\ f_2(\vec{x}, \vec{u}) \\ \vdots \\ f_n(\vec{x}, \vec{u}) \end{bmatrix}, \tag{17}$$

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 from the earlier section to approximate

$$f_i(\vec{x}, \vec{u}) \approx f_i(\vec{x}^{\star}, \vec{u}^{\star}) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}(x^{\star}, u^{\star}) \cdot (x_j - x_j^{\star}) + \sum_{j=1}^{k} \frac{\partial f_i}{\partial u_j}(x^{\star}, u^{\star}) \cdot (u_j - u_j^{\star}). \tag{18}$$

We can rearrange the above linearization as the matrix multiplication

$$f_i(\vec{x}, \vec{u}) \approx f_i(\vec{x}^{\star}, \vec{u}^{\star}) \tag{19}$$
$$+ \begin{bmatrix} \frac{\partial f_i}{\partial x_1}(\vec{x}^{\star}, \vec{u}^{\star}) & \cdots & \frac{\partial f_i}{\partial x_n}(\vec{x}^{\star}, \vec{u}^{\star}) \end{bmatrix} \cdot (\vec{x} - \vec{x}^{\star}) \tag{20}$$
$$+ \begin{bmatrix} \frac{\partial f_i}{\partial u_1}(\vec{x}^{\star}, u^{\star}) & \cdots & \frac{\partial f_i}{\partial u_k}(\vec{x}^{\star}, u^{\star}) \end{bmatrix} \cdot (\vec{u} - \vec{u}^{\star}). \tag{21}$$

The two row vectors above can be thought of as the derivatives[5] of $f_i(\vec{x}, \vec{u})$ with respect to $\vec{x}$ and $\vec{u}$, so we denote these rows by $\frac{\partial f_i}{\partial \vec{x}}$ and $\frac{\partial f_i}{\partial \vec{u}}$. These are row-vector-valued functions of column-vectors $\vec{x}^{\star}, \vec{u}^{\star}$, in the same way that a derivative of a scalar-valued function of scalars is itself a scalar-valued function. So,

$$f_i(\vec{x}, \vec{u}) \approx f_i(\vec{x}^{\star}, \vec{u}^{\star}) + \frac{\partial f_i}{\partial \vec{x}}(\vec{x}^{\star}, \vec{u}^{\star}) \cdot (\vec{x} - \vec{x}^{\star}) + \frac{\partial f_i}{\partial \vec{u}}(\vec{x}^{\star}, \vec{u}^{\star}) \cdot (\vec{u} - \vec{u}^{\star}) \tag{22}$$

This was the linearization for just one element of $\vec{f}$, so we can now stack the above linearization for all f_i to obtain a linearization for the original vector-valued $\vec{f}(\vec{x}, \vec{u})$:

$$\vec{f}(\vec{x}, \vec{u}) \approx \vec{f}(\vec{x}^{\star}, \vec{u}^{\star}) \tag{23}$$

[5]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 a column vector and is the transpose of the derivative of the function f.

$$+ \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\vec{x}^\star, \vec{u}^\star) & \cdots & \frac{\partial f_1}{\partial x_n}(\vec{x}^\star, \vec{u}^\star) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\vec{x}^\star, \vec{u}^\star) & \cdots & \frac{\partial f_n}{\partial x_n}(\vec{x}^\star, \vec{u}^\star) \end{bmatrix} \cdot (\vec{x} - \vec{x}^\star) \tag{24}$$

$$+ \begin{bmatrix} \frac{\partial f_1}{\partial u_1}(\vec{x}^\star, \vec{u}^\star) & \cdots & \frac{\partial f_1}{\partial u_k}(\vec{x}^\star, \vec{u}^\star) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1}(\vec{x}^\star, \vec{u}^\star) & \cdots & \frac{\partial f_n}{\partial u_k}(\vec{x}^\star, \vec{u}^\star) \end{bmatrix} \cdot (\vec{u} - \vec{u}^\star), \tag{25}$$

We call these two matrices of partial derivatives the *Jacobian*(s) of $\vec{f}$ with respect to $\vec{x}$ and $\vec{u}$.

Formally, the Jacobian is the partial derivative of a vector-valued function $\vec{f}$ with respect to a vector input $\vec{x}$. Thus, in analogy with our previous notations, we may write the Jacobian as $\frac{\partial \vec{f}}{\partial \vec{x}}$. Other notations include, for example, $J_{\vec{x}}\vec{f}$. These are matrix-valued functions, just as the derivative of a scalar function by a scalar variable is a scalar function.

Here they are evaluated at $(\vec{x}^\star, \vec{u}^\star)$. This gives us

$$\vec{f}(\vec{x}, \vec{u}) \approx \vec{f}(\vec{x}^\star, \vec{u}^\star) + \frac{\partial \vec{f}}{\partial \vec{x}}(\vec{x}^\star, \vec{u}^\star) \cdot (\vec{x} - \vec{x}^\star) + \frac{\partial \vec{f}}{\partial \vec{u}}(\vec{x}^\star, \vec{u}^\star) \cdot (\vec{u} - \vec{u}^\star) \tag{26}$$

so we have solved for $A = \frac{\partial \vec{f}}{\partial \vec{x}}(\vec{x}^\star, \vec{u}^\star)$ and $B = \frac{\partial \vec{f}}{\partial \vec{u}}(\vec{x}^\star, \vec{u}^\star)$ in our desired linear model.

Note that this formula is exactly how our other linearization formulas look.

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 Equilibrium Points

Now, let's see how this linearization plugs back into our original differential equation. We see that

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) = f(\vec{x}, \vec{u}) = \vec{f}(\vec{x}^\star, \vec{u}^\star) + \frac{\partial \vec{f}}{\partial \vec{x}}(\vec{x}^\star, \vec{u}^\star) \cdot (\vec{x}(t) - \vec{x}^\star) + \frac{\partial \vec{f}}{\partial \vec{u}}(\vec{x}^\star, \vec{u}^\star) \cdot (\vec{u}(t) - \vec{u}^\star) + \vec{w}(t). \tag{27}$$

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 $\vec{x}(t) - \vec{x}^\star$ on the right, but $\vec{x}(t)$ on the left. To fix this, we will define new state variables $\Delta\vec{x}(t) = \vec{x}(t) - \vec{x}^\star$ and new control inputs $\Delta\vec{u}(t) = \vec{u}(t) - \vec{u}^\star$.

By rewriting $\vec{x}(t) = \Delta\vec{x}(t) + \vec{x}^\star$, we have

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\big(\vec{x}^\star + \Delta\vec{x}(t)\big) = \frac{\mathrm{d}}{\mathrm{d}t}\vec{x}^\star(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Delta\vec{x}(t). \tag{28}$$

Since $\vec{x}^\star$ is a constant for all time, we know that $\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}^\star = \vec{0}$. Hence

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\Delta\vec{x}(t). \tag{29}$$

Expanding out the derivative, we obtain

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\Delta\vec{x}(t) &= \frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) && (30)\\ &= \vec{f}(\vec{x}^\star,\vec{u}^\star) + \frac{\partial \vec{f}}{\partial \vec{x}}(\vec{x}^\star,\vec{u}^\star)\cdot(\vec{x}(t)-\vec{x}^\star) + \frac{\partial \vec{f}}{\partial \vec{u}}(\vec{x}^\star,\vec{u}^\star)\cdot(\vec{u}(t)-\vec{u}^\star) + \vec{w}(t) && (31)\\ &= \vec{f}(\vec{x}^\star,\vec{u}^\star) + \frac{\partial \vec{f}}{\partial \vec{x}}(\vec{x}^\star,\vec{u}^\star)\cdot\Delta\vec{x}(t) + \frac{\partial \vec{f}}{\partial \vec{u}}(\vec{x}^\star,\vec{u}^\star)\cdot\Delta\vec{u}(t) + \vec{w}(t). && (32)\end{aligned}$$

Then the final term to remove to get a linear model is the $\vec{f}(\vec{x}^\star,\vec{u}^\star)$ on the right side of the equation. Notice that this term depends only on the choice of expansion point. Thus, when we linearize a system, it is our job to pick expansion points such that

$$\vec{f}(\vec{x}^\star,\vec{u}^\star) = \vec{0}. \tag{33}$$

Such points $(\vec{x}^\star,\vec{u}^\star)$ are called *equilibrium points* of $\vec{f}$. When we linearize a dynamical system around such an equilibrium, the approximation is now a linear system:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\Delta\vec{x}(t) &= \frac{\partial \vec{f}}{\partial \vec{x}}(\vec{x}^\star,\vec{u}^\star)\cdot\Delta\vec{x}(t) + \frac{\partial \vec{f}}{\partial \vec{u}}(\vec{x}^\star,\vec{u}^\star)\cdot\Delta\vec{u}(t) + \vec{w}(t) && (34)\\ &= A\cdot\Delta\vec{x}(t) + B\cdot\Delta\vec{u}(t) + \vec{w}(t) && (35)\end{aligned}$$

where now $A = \frac{\partial \vec{f}}{\partial \vec{x}}(\vec{x}^\star,\vec{u}^\star)$ and $B = \frac{\partial \vec{f}}{\partial \vec{u}}(\vec{x}^\star,\vec{u}^\star)$.

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 linearized system about the chosen operating point. Note that this feedback control will drive $\Delta\vec{x} \to 0$ which means it drives $\vec{x} \to \vec{x}^\star$. In this way, we can think of $\vec{x}^\star$ as our desired value of $\vec{x}(t)$. It is also important to remember that the actual control applied will be $\vec{u}(t) = \vec{u}^\star + K(\vec{x}(t) - \vec{x}^\star)$ — because the K has been calculated for the linearized system and we need to apply a control in the original nonlinear system.

In general, there are are many potential solutions for an equilibrium point. 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 $\vec{x}^\star$. In that case, you will want to solve for the the corresponding $\vec{u}^\star$ that satisfies (33) with that particular state $\vec{x}^\star$. 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[6] or graphical[7] methods to solve the nonlinear system of equations (33).

The ability to linearize is what makes linear control practical since many (if not most) real-world systems that we encounter are fundamentally nonlinear. The linearized system of differential equations can also be

[6]So, how do you solve nonlinear systems of equations like (33)? This can be challenging at times but fortunately software packages exist for you to use. For example, SciPy has `scipy.optimize.fsolve` for you. Alternatively, you can use the Gauss-Newthon method.

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

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.

But what if we start in discrete-time? In general, it is not valid to linearize in discrete-time, since the sampling interval is fixed in discrete time and is probably larger than the interval in which our linearization is a useful approximation. Thus we generally need to start from continuous-time systems and linearize to discrete-time systems, or else go from a discrete-time system to a continuous-time system by taking the limit as the sampling interval goes to 0, linearize, and convert back by discretization.

5 Tracking 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, we can similarly consider what happens when $\vec{x}^\star$ is no longer constant, and so we want $\vec{x}(t)$ to follow some desired trajectory $\vec{x}^\star(t)$. Then to get the same linear system as (34), we will want operating points such that

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}^\star(t) = \vec{f}(\vec{x}^\star(t), \vec{u}^\star(t)). \tag{36}$$

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 equilibrium 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.

6 Second-Order Approximation

There is one final loose end to tie up; namely, what happens if we take more than just the linear term in our truncated Taylor series approximation? It turns out that going more than two terms, even for scalar-valued functions of vector-arguments, gets very messy. We would need more advanced versions of the tools we have already covered, along with a deeper understanding of abstract linear algebra and the geometry of vector spaces, to efficiently tackle such problems, which form the backbone of so-called *tensor calculus* (tensors being essentially n-dimensional collections of numbers, in the sense that vectors are when $n = 1$ and matrices are when $n = 2$). However, there is one setting where going beyond linearization makes sense to us right now: when we are approximating a scalar function of a vector input by using up to two derivatives.

Namely, suppose $f(\vec{x})$ is a scalar-valued function of a vector $\vec{x} \in \mathbb{R}^n$. We would like to approximate f around a point $\vec{x}^\star$ using the first and second derivatives of f.

The first derivative of f, $\frac{\partial f}{\partial \vec{x}}$, is something we already know how to calculate from earlier in this note. But what about the second derivative? As before, let us retreat to the single-variable case and then do a concrete example to gain intuition, redoing our previous derivation with the quadratic term in mind.

First, let us consider the scalar case. Suppose f is a scalar function of a scalar variable x. Then Taylor's theorem tells us

$$\begin{aligned} f(x) &= f(x^\star) + \frac{\mathrm{d}f}{\mathrm{d}x}(x^\star) + \frac{1}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x^\star) + \cdots && (37) \\ &\approx f(x^\star) + \frac{\mathrm{d}f}{\mathrm{d}x}(x^\star) + \frac{1}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x^\star). && (38) \end{aligned}$$

Note that the second derivative has a coefficient of $\frac{1}{2}$ in front of it.

Now we do an example. Let us suppose we have the system

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = f(x(t), u(t)) \tag{39}$$

where $f(x, u) = x^3u^2$ as before. Again, we perturb both $x = x^\star + \delta x$ and $u = u^\star + \delta u$. We write

$$\begin{aligned} f(x^\star + \delta x, u^\star + \delta u) &= (x^\star + \delta x)^3(u^\star + \delta u)^2 && (40) \\ &\approx x^{\star 3}u^{\star 2} + 3x^{\star 2}u^{\star 2} \cdot \delta x + 2x^{\star 3}u^\star \cdot \delta u && (41) \\ &\quad + 3x^\star u^{\star 2} \cdot (\delta x)^2 + x^{\star 3} \cdot (\delta u)^2 + 6x^{\star 2}u^\star \cdot (\delta x)(\delta u). && (42) \end{aligned}$$

Here, in order to carry through a quadratic approximation in our variables δx, δu, we kept all the linear and quadratic terms in the expansion, and not just the linear terms like we did for the linear approximation. Here, note that quadratic doesn't necessarily mean that *one* variable has to have a power of two – just that the sum of powers of the variables for that term is equal to two, as seen in the last term where the powers on δx and δu are both 1. This definition is consistent with other contexts in which we find multivariate polynomials.

The key insight to continue from here is that we can extend our concept of partial derivatives to get second-order partial derivatives, and indeed n^{th} order partial derivatives. Indeed, one can write

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right) \qquad \frac{\partial^2 f}{\partial u^2} = \frac{\partial}{\partial u}\left(\frac{\partial f}{\partial u}\right) \qquad \frac{\partial^2 f}{\partial x \partial u} = \frac{\partial}{\partial u}\left(\frac{\partial f}{\partial x}\right), \tag{43}$$

and so on. It is a fact of multivariable calculus that under mild conditions, such as the second derivatives being continuous at the point of evaluation, one can interchange the order of taking partial derivatives, i.e., $\frac{\partial^2 f}{\partial x \partial u} = \frac{\partial^2 f}{\partial u \partial x}$. The precise result is called Clairaut's theorem.

Rewriting our earlier approximation in terms of the first and second partial derivatives, we can write

$$f(x^\star + \delta x, u^\star + \delta u) \approx f(x^\star, u^\star) + \frac{\partial f}{\partial x}(x^\star, u^\star) \cdot \delta x + \frac{\partial f}{\partial x}(x^\star, u^\star) \cdot \delta u \tag{44}$$

$$+ \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x^\star, u^\star) \cdot (\delta x)^2 + \frac{1}{2}\frac{\partial^2 f}{\partial u^2}(x^\star, u^\star) \cdot (\delta u)^2 + \frac{\partial^2 f}{\partial x \partial u}(x^\star, u^\star) \cdot (\delta x)(\delta u). \tag{45}$$

This is very close to symmetric in all the partial derivatives, with one exception: the so-called "mixed partial" $\frac{\partial^2 f}{\partial x \partial u}$ has coefficient 1 while all other partial derivatives have coefficient $\frac{1}{2}$ (similarly to the single-variable case), and there is no partial $\frac{\partial^2 f}{\partial u \partial x}$.

We can change both of those at the same time, by using Clairaut's theorem to interchange the partial derivatives. Since $\frac{\partial^2 f}{\partial x \partial u} = \frac{\partial^2 f}{\partial u \partial x}$, we can write $\frac{\partial^2 f}{\partial x \partial u} = \frac{1}{2}\frac{\partial^2 f}{\partial x \partial u} + \frac{1}{2}\frac{\partial^2 f}{\partial u \partial x}$, and obtain the much nicer symmetric expression

$$f(x^\star + \delta x, u^\star + \delta u) \approx f(x^\star, u^\star) + \frac{\partial f}{\partial x}(x^\star, u^\star) \cdot \delta x + \frac{\partial f}{\partial x}(x^\star, u^\star) \cdot \delta u \tag{46}$$

$$+ \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x^\star, u^\star) \cdot (\delta x)^2 + \frac{1}{2}\frac{\partial^2 f}{\partial u^2}(x^\star, u^\star) \cdot (\delta u)^2 \tag{47}$$

$$+ \frac{1}{2}\frac{\partial^2 f}{\partial x \partial u}(x^\star, u^\star) \cdot (\delta x)(\delta u) + \frac{1}{2}\frac{\partial^2 f}{\partial u \partial x}(x^\star, u^\star) \cdot (\delta u)(\delta x). \tag{48}$$

In fact this formula holds in general if we have a general analytic scalar-valued function f in two variables. Writing it in a similar fashion to the previous equations, we have

$$f(x, u) \approx f(x^\star, u^\star) + \frac{\partial f}{\partial x}(x^\star, u^\star) \cdot (x - x^\star) + \frac{\partial f}{\partial x}(x^\star, u^\star) \cdot (u - u^\star) \tag{49}$$

$$+ \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x^\star, u^\star) \cdot (x - x^\star)^2 + \frac{1}{2}\frac{\partial^2 f}{\partial u^2}(x^\star, u^\star) \cdot (u - u^\star)^2 \tag{50}$$

$$+ \frac{1}{2}\frac{\partial^2 f}{\partial x \partial u}(x^\star, u^\star) \cdot (x - x^\star)(u - u^\star) + \frac{1}{2}\frac{\partial^2 f}{\partial u \partial x}(x^\star, u^\star) \cdot (u - u^\star)(x - x^\star). \tag{51}$$

It even generalizes if we have a general analytic scalar-valued function f of a vector $\vec{x} \in \mathbb{R}^n$:

$$f(\vec{x}) = f(\vec{x}^\star) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\vec{x}^\star) \cdot (x_i - x_i^\star) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j}(\vec{x}^\star) \cdot (x_i - x_i^\star)(x_j - x_j^\star) \tag{52}$$

Remember that when we derived linear approximations, we developed a nice vector-based expression for the terms involving the first derivatives:

$$\sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\vec{x}^\star) \cdot (x_i - x_i^\star) = \frac{\partial f}{\partial \vec{x}}(\vec{x}^\star) \cdot (\vec{x} - \vec{x}^\star). \tag{53}$$

Now we want to come up with a nice, idiomatic expression for the second derivatives:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j}(\vec{x}^\star) \cdot (x_i - x_i^\star)(x_j - x_j^\star) = ? \tag{54}$$

To do this in the linear case, we defined a new object – the row vector of first partial derivatives $\frac{\partial f}{\partial \vec{x}}$, such that $\frac{\partial f}{\partial \vec{x}}(\vec{x}^\star) \cdot (\vec{x} - \vec{x}^\star)$ gave us a scalar.

Now we would like to get a scalar from some product of $\vec{x} - \vec{x}^\star$ and some function of the second derivatives. One guess we can make is to go entirely based off object types, in the following manner.

One sees that a matrix times a column vector is a column vector, and that a row vector times a column vector is a scalar. So a row vector times a matrix times a column vector should be a scalar. Let us naively guess that the matrix should contain all our second partial derivatives, and that our column and row vectors should be $\vec{x} - \vec{x}^\star$, respectively, since they seem like the only candidate quantities we can work with.

This actually turns out to work. More specifically, define the *Hessian* of f to be the matrix containing all its second partial derivatives:

$$\frac{\partial^2 f}{\partial \vec{x}^2} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{55}$$

We can also use some other notations like $H_{\vec{x}} f$ that you might see in discussion. One thing to note here is that the *Jacobian*, i.e., $\frac{\partial \vec{f}}{\partial \vec{x}}$, the first derivatives of a vector-valued function of a vector, is different from the *Hessian*, i.e., $\frac{\partial^2 f}{\partial \vec{x}^2}$, the second derivative of a scalar-valued function of a vector. But they are connected in the following way: if f is a scalar-valued function of a vector, and $\vec{g}(\vec{x}) = \left(\frac{\partial f}{\partial \vec{x}}(\vec{x})\right)^\top$ (so that g is a column vector), then $\frac{\partial \vec{g}}{\partial \vec{x}} = \frac{\partial^2 f}{\partial \vec{x}^2}$. This is analogous to the way one takes second derivatives by taking the derivative of the first derivative, and so on.

Back to our guess. We plug in and try to compute:

$$(\vec{x} - \vec{x}^\star)^\top \left(\frac{\partial^2 f}{\partial \vec{x}^2}(\vec{x}^\star)\right)(\vec{x} - \vec{x}^\star) = \sum_{i=1}^{n}\sum_{j=1}^{n} \left(\frac{\partial^2 f}{\partial \vec{x}^2}(\vec{x}^\star)\right)_{ij} (\vec{x} - \vec{x}^\star)_i (\vec{x} - \vec{x}^\star)_j \tag{56}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j}(\vec{x}^\star)(x_i - x_i^\star)(x_j - x_j^\star). \tag{57}$$

This turns out to be exactly the quantity we need! So we have found out how to simplify our original expression for the second derivatives:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j}(\vec{x}^\star) \cdot (x_i - x_i^\star)(x_j - x_j^\star) = (\vec{x} - \vec{x}^\star)^\top \left(\frac{\partial^2 f}{\partial \vec{x}^2}(\vec{x}^\star)\right)(\vec{x} - \vec{x}^\star). \tag{58}$$

Through this, we can simplify our expression for quadratic approximation:

$$f(\vec{x}) \approx f(\vec{x}^\star) + \frac{\partial f}{\partial \vec{x}}(\vec{x}^\star)(\vec{x} - \vec{x}^\star) + \frac{1}{2}(\vec{x} - \vec{x}^\star)^\top \left(\frac{\partial^2 f}{\partial \vec{x}^2}(\vec{x}^\star)\right)(\vec{x} - \vec{x}^\star). \tag{59}$$

Quadratic approximation will be useful to us in the next few sections on classification and machine learning, where sometimes linearization is not helpful to us because our functions have some kind of "essential non-linearity" that must be taken into account during analysis. However, linearization works just fine for many applications, such as robotics, control, circuits, and so on.

Contributors:

- Rahul Arya.
- Anant Sahai.
- Ashwin Vangipuram.
- Druv Pai.