University of California, Berkeley

ME/EECS/BioE C106B

Robotic Manipulation & Interaction

Massimiliano de Sa

These notes were written for the UC Berkeley course ME/EECS/BioE C106B in the Spring 2023 semester. I hope you find them useful!
They are primarily based off of *A Mathematical Introduction to Robotic Manipulation* by Murray, Li, and Sastry, *Nonlinear Systems: Analysis, Stability, and Control* by Sastry, and *Feedback Systems: An Introduction for Scientists and Engineers* by Astrom and Murray, and have been designed to closely follow the 106B course lectures in both ordering and content.
If you have any questions, please reach out to me at mz.desa at berkeley.edu.

Contents

Chapter 1

Dynamical Systems

Welcome to 106B! In this course, we'll discuss concepts ranging from dynamics to feedback control, path planning, vision, and beyond! We'll build upon the foundations of dynamics and control we built in 106A to form a comprehensive yet deep overview of several key fields in robotics.

1.1 Modeling Dynamical Systems

In this section, we'll begin our discussion of dynamical systems, systems who evolve with the passage of time. We'll consider different methods of describing these systems, and see how differential equations provide us with the mathematical tools we need for their analysis. Let's get started!

1.1.1 Classifying Dynamical Systems

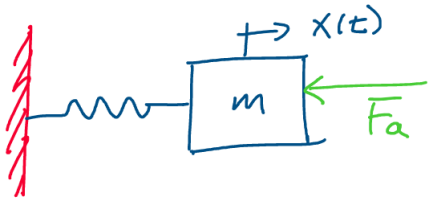
Nonlinear Systems

How can we describe the evolution of a system in time? As a robot arm moves, for example, we know that torques and forces acting on the arm govern its path through space. How can we mathematically describe the relationship between these forces and the eventual trajectory of the arm?

Using the language of *differential equations*, we may describe how systems such as robot arms, drones, autonomous vehicles, and more move with the passage of time!

As we progress through the course, we'll find it exceedingly useful to have *general* descriptions for dynamical systems. This will enable us to explore results that hold not just for one system but for *arbitrary* systems. To achieve this generality in our results, it's important that we come up with certain conventions of how to write out the differential equations for these systems.

Let's write out the equations of motion for a few systems and see if there are any common elements that stand out. Let's begin with a simple system: a mass that oscillates on a spring in the presence of an applied force, F_a:

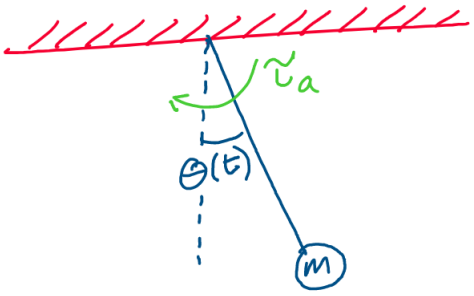


Above: A mass oscillates on a spring with an applied force, F_a.

From our knowledge of dynamics, we know that the equations of motion of this system are given by the differential equation:

$$m\ddot{x} = -kx - F_a \tag{1.1}$$

Where x is the position of the mass, $\ddot{x}$ is the acceleration of the mass, k is the spring constant, and F_a is the applied force.
Note that we use dot notation for convenience when expressing the time derivative of a variable. Recall that the number of dots above a variable is the number of time derivatives that have been taken. For instance, $\ddot{x}$ refers to the second time derivative of x.
Let's think about another physical system, a mass that swings on a pendulum around a fixed point with an applied torque τ_a. As we write its equations of motion, see if there any any common themes you can pick out between the first two systems.

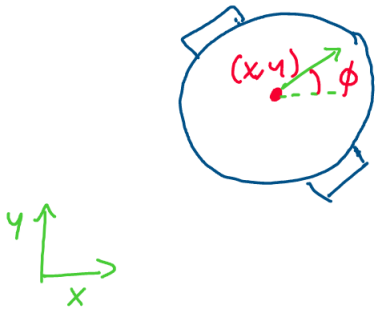


Above: A pendulum of mass m swings around a fixed point

If this pendulum has a length l, mass m, and angular position θ, its dynamics may be described by the differential equation:

$$ml^2\ddot{\theta} = -mg\sin\theta - \tau_a \tag{1.2}$$

Finally, let's consider a third system, a turtlebot! Instead of describing the equations of motion of the turtlebot using methods of dynamics, we can make use of *kinematic constraints* to arrive at the governing differential equations. We'll perform an in-depth exploration of how this is done in the coming sections.



Above: A turtlebot traveling in an (x, y) coordinate frame

We may show that the kinematics of this turtlebot are described by the following equations of motion:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v\cos\phi \\ v\sin\phi \\ \omega \end{bmatrix} \tag{1.3}$$

Where (x, y, ϕ) are the coordinates and orientation of the turtlebot with respect to a world coordinate frame and v and ω are the speed and angular steering rate of the turtlebot, which may be controlled by a user.

What similarities do these three dynamical systems share? Firstly, something that we notice is that all of the systems have a set of variables that describe the *position* or *velocity* of the system at any given point! For the mass-spring system, this was the x position of the mass, for the pendulum the angle θ, and for the turtlebot the coordinates (x, y) and orientation ϕ.

Secondly, we observe that each system has some sort of input! For the mass, there was an applied force, F_a, for the pendulum an applied torque, τ_a, and for the turtlebot a velocity v and angular steering rate ω.

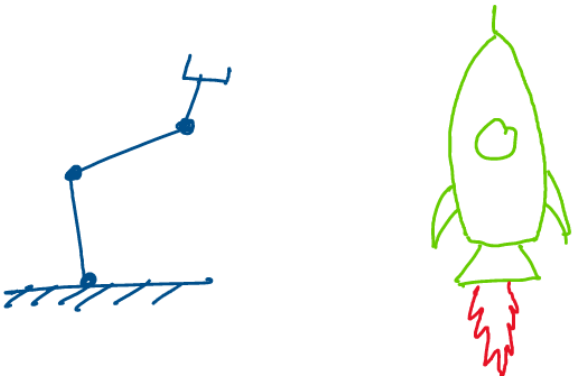
With these thoughts in mind, we may introduce the standard form for a non-linear system, known as the **state space representation**:

$$\dot{x} = f(x, u), x \in \mathbb{R}^n, u \in \mathbb{R}^m \qquad \textit{State Equation} \tag{1.4}$$

$$y = h(x, u),\ y \in \mathbb{R}^p \qquad \textit{Output Equation} \tag{1.5}$$

The first equation, $\dot{x} = f(x, u)$ is known as the **state equation**. This equation *completely* describes how the system will move as time passes, and encodes all of the information about the physical laws and processes governing the system. If the state equation $f(x, u)$ is not an explicit function of time, the system is said to be **time invariant**. If the state equation *is* an explicit function of time, the state equation is written by convention as $f(x, u, t)$ and the system is said to be **time variant**.

Most of the robotic systems we'll study are time invariant, as the governing dynamics of systems such as robot arms typically do not change as time passes. A system such as a rocket, however, whose mass changes with time as fuel is burned, would be considered time variant.



Above: Robotic systems are typically time invariant, as the physical properties of a robot arm do not change in time. A rocket is time variant, as it loses mass by burning fuel as time passes.

The vector $x \in \mathbb{R}^n$ is known as the **state vector** of the system. It contains smallest possible set of variables necessary to describe the configuration of the system at any given point. For the turtlebot, for example, the state vector would be:

$$x = \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} \in \mathbb{R}^3 \tag{1.6}$$

Notice that we *cannot* control the states of the system directly!
The **input vector** $u \in \mathbb{R}^m$, on the other hand, is a vector of all variables that we have *complete* control over! We can change the elements of the input vector at our will to influence the behavior of the system. For the turtlebot, for example, we can completely control the velocity and steering rate of the vehicle to drive it to a particular location!
The input vector of the turtlebot would be:

$$u = \begin{bmatrix} v \\ \omega \end{bmatrix} \in \mathbb{R}^2 \tag{1.7}$$

What role does the second equation, $y = h(x, u)$, play in the state space representation? $y = h(x, u)$ is known as the **output equation**. The output equation has *no effect* on the actual dynamics of the system! We can pick the function $h(x, u)$ to be *anything* we want. The **output vector** of the system, $y \in \mathbb{R}^p$, typically contains our variables of interest.
For example, in the case of the turtlebot, we may be interested in controlling the x and y position of the turtlebot as it moves through the environment, but not as interested in the orientation, ϕ. In this case, we could define the output of the system to be:

$$y = \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2 \tag{1.8}$$

The output of a system is typically a function of the state vector but can also be a function of the input vector in some cases, hence the inclusion of u in $y = h(x, u)$.
In addition to being variables of interest, the outputs of a system are also commonly chosen to be the states of the system that are observable from different sensors. This choice of output is particularly common when designing filters to determine the state of a system.
Let's think critically for a moment about the state space description of a system. What challenges do we notice with this representation?

$$\dot{x} = f(x, u), \ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{1.9}$$
$$y = h(x, u), \ y \in \mathbb{R}^p \tag{1.10}$$

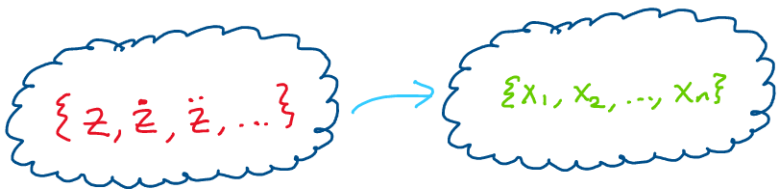
A potential problem that we notice with this formulation is that this is a *first order* system of differential equations. When we take a closer look at the dynamics equations for the mass-spring and pendulum systems, however, what do we observe?

$$m\ddot{x} = -kx - F_a \tag{1.11}$$
$$ml^2\ddot{\theta} = -mgl\sin\theta - \tau_a \tag{1.12}$$

Although the state space representation of a system, $\dot{x} = f(x, u), \ y = h(x, u)$ is a *first order* system of differential equations, both of these equations are second order, due to the appearance of a second derivative with respect to time! Does this mean that our general state space representation *fails* to describe all systems, or is there a way we can *transform* higher order systems to first order systems with equivalent behavior?
Let's see if we can convert these higher order nonlinear systems into a system of first order differential equations. Suppose that we have an n^{th} order, nonlinear differential equation, described by the following equation:

$$z^{(n)} = \frac{d^n z}{dt^n} = h(z, u) \tag{1.13}$$

Where $z \in \mathbb{R}$ is a scalar, $u \in \mathbb{R}^m$ is an input vector, and $h(z, u)$ is a function describing the dynamics of the system. Note that $z^{(n)}$ is shorthand for the n^{th} time derivative of z.
Let's see if we can find a *magic* change of variables from z and its derivatives to a new set, $\{x_1, x_2, ..., x_n\}$, that *transform* this system from an n^{th} order differential equation to a *system* of n first order differential equations!



Consider the following choice of variables:

$$x_1 = z \tag{1.14}$$

$$x_2 = \frac{dz}{dt} \tag{1.15}$$

$$x_3 = \frac{d^2z}{dt^2} \tag{1.16}$$

$$\vdots \tag{1.17}$$

$$x_n = \frac{d^{n-1}z}{dt^{n-1}} \tag{1.18}$$

We can put these x_i together into a vector, which we call x:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \tag{1.19}$$

Does this choice of variables allow us to *rewrite* our original system as $\dot{x} = f(x, u)$? Let's take the first derivative of each x_i and see what happens! For $1 \leq i < n$, we find:

$$\dot{x}_1 = \frac{dz}{dt} = x_2 \tag{1.20}$$

$$\dot{x}_2 = \frac{d^2z}{dt^2} = x_3 \tag{1.21}$$

$$\vdots \tag{1.22}$$

$$\dot{x}_{n-1} = \frac{d^{n-1}z}{dt^{n-1}} = x_n \tag{1.23}$$

When we take the first time derivative of each x_i, for $1 \leq i < n$, we get x_{i+1}! What happens when we take the time derivative of x_n?

$$\dot{x}_n = \frac{d}{dt}(z^{(n-1)}) = z^{(n)} = h(z, u) \tag{1.24}$$

Thus, when we take first time derivative of x_n, we get our original differential equation, $h(z, u) = h(x_1, u)$! Thus, by choosing these variables, we have preserved the dynamics of the system and converted an n^{th} order differential equation into a system of n first order differential equations.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ h(x_1, u) \end{bmatrix} \tag{1.25}$$

We have now successfully converted our original n^{th} order differential equation into a system of n first order differential equations in state space form:

$$\dot{x} = f(x, u),\ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{1.26}$$

The choice of variables $\{x_1, x_2, ..., x_n\} = \{z, \dot{z}, ..., z^{(n-1)}\}$ that we used to accomplish this transformation are known as **phase variables**.
Let's gain some practice converting higher order systems into state space using phase variables. Recall the dynamics of the mass-spring system we discussed above:

$$m\ddot{x} = -kx - F_a \tag{1.27}$$

We notice that this system is *second order*. Thus, we need two phase variables, since we will convert this second order equation to a system of *two* first order equations. Using the phase variable convention, we choose:

$$x_1 = x,\ x_2 = \dot{x} \tag{1.28}$$

Since F_a is an applied force that we can control, we define an input vector $u = F_a$. Now, we rewrite the system in terms of these variables:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{-k}{m}x_1 - \frac{1}{m}u \end{bmatrix} \tag{1.29}$$

Thus, we have successfully rewritten our second order system in the form:

$$\dot{x} = f(x, u),\ x = [x_1, x_2] = [x, \dot{x}],\ u = F_a \tag{1.30}$$

Note that because x is a variable that is commonly used in the description of physical systems, the variable q is often used in the place of x as the state vector of the system to avoid confusion.
As a general rule of thumb for checking your choice of phase variables, the highest derivative value in your choice of phase variables should be *one lower* than the order of your original differential equation. For instance, in the example above, our original differential equation was second order, and the highest derivative phase variable, $\dot{x}$, was first order. This comes from the fact that in state space, we always take the first time derivative of our entire state vector.

Control Affine Systems

Now that we have a general form for describing nonlinear systems, we can ask the question: are there certain *subclasses* of nonlinear systems that are easier to work with?
As we'll soon discover in our study of feedback control, designing a general feedback controller to control the state vector of an arbitrary nonlinear system:

$$\dot{x} = f(x, u) \tag{1.31}$$

Is quite complex! Because of the ambiguous nature of the function f - that f can be almost *any* function in the general nonlinear case - it's difficult to design controllers for the most general class of nonlinear systems!
Certain subsets of the class of nonlinear systems, however, have the advantage of being particularly common in practice and much simpler to design feedback controllers for! One such class is the set of **control affine** nonlinear systems. These are nonlinear systems whose state equation may be written in the form:

$$\dot{x} = f(x) + g(x)u,\ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{1.32}$$

Now, instead of having the input and state dynamics combined into a single function, $\dot{x} = f(x, u)$, the input dynamics may be *separated* from those purely related to the state vector! Let's break down the different elements of a control-affine system.
The function $f(x)$ is known as the **drift dynamics** of the system. These are the dynamics that have *nothing* to do with the inputs to the system, and are purely a function of the state vector. $f(x) : \mathbb{R}^n \to \mathbb{R}^n$ is a vector-valued function:

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix} \in \mathbb{R}^n,\ f_i(x) : \mathbb{R}^n \to \mathbb{R}, 1 \leq i \leq n \tag{1.33}$$

Where each f_i within the vector $f(x)$ is a scalar-valued function. If the input u to the system were zero, the state vector would *drift* according to the value of the function $f(x)$.
In the input term, $g(x) : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ is a matrix-valued function of x that determines the effect of the input on the evolution of the state vector. Each element of $g(x)$ is a scalar function:

$$g(x) = \begin{bmatrix} g_{11}(x) & \dots & g_{1m}(x) \\ \vdots & \ddots & \vdots \\ g_{n1}(x) & \dots & g_{nm}(x) \end{bmatrix} \in \mathbb{R}^{n \times m} \tag{1.34}$$

Because of they separate the control input from the rest of the system dynamics, control affine systems prove to be much easier to work with when designing feedback controllers. Although not every nonlinear system is control affine, a sizeable number of complex robotic systems are, making these systems of key importance to us.
What does the term affine actually mean? In general, a function $h(x) : \mathbb{R}^n \to \mathbb{R}^n$ is said to be *affine in* x if it can be expressed:

$$h(x) = Ax + b \tag{1.35}$$

Where A is a matrix and b is a vector. Since a control affine system is of the form:

$$\dot{x} = f(x) + g(x)u \tag{1.36}$$

It is considered to be *affine* in the control input, u. This property gives the class of control affine systems its name!

Linear Systems

Control affine systems provide a significant reduction in challenges with respect to full nonlinear systems, $\dot{x} = f(x, u)$. Are there any subclasses of dynamical systems that can further simplify our analyses?
The class of **linear systems** is perhaps the most simple yet one of the most useful class of dynamical systems. Linear systems are conventionally written in the form:

$$\dot{x} = Ax + Bu,\ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{1.37}$$

$$y = Cx + Du,\ y \in \mathbb{R}^p \tag{1.38}$$

If A, B, C, D are constant matrices that don't change as time passes, the system is said to be **linear time invariant** (LTI). If these matrices *do* change in time, they are written as $A(t), B(t), C(t), D(t)$, and the system is said to be **linear time variant** (LTV).
Inspecting the structure of the equations, we have the familiar setup of a state equation and output equation expressing the dynamics of the system. However, in this case, both the state vector, x, and input vector, u, are simply scaled by matrices instead of being manipulated by various nonlinear functions.
Since such a representation allows us to use the elegant language of linear algebra to perform system analysis, the class of linear differential equations proves to be exceptionally convenient in robotics and control.
As we'll see shortly, we can make remarkably deep conclusions about many systems by inspecting linear algebraic objects such as eigenvalues and eigenvectors.
Mathematically, what does it mean for a system to be linear?
If, u_1, u_2 are arbitrary inputs, α, β are scalar constants, and $y(t, x_0, u)$ is the output of a system at time t starting at initial condition x_0 with an input u, the following three properties must be satisfied for a system to be considered **input-output linear**:

1. $y(t, \alpha x_1 + \beta x_2, 0) = \alpha y(t, x_1, 0) + \beta y(t, x_2, 0)$
2. $y(t, \alpha x_1, \beta u) = \alpha y(t, x_1) + \beta y(t, 0, u)$
3. $y(t, 0, \alpha u_1 + \beta u_2) = \alpha y(t, 0, u_1) + \beta y(t, 0, u_2)$

These three conditions express the linearity of the output with respect to the initial conditions and inputs to the system. For the linear system, $\dot{x} = Ax + Bu, y = Cx + Du$, we can verify these facts using the properties of matrix algebra.

1.1.2 Solving Differential Equations

Now that we have established conventions for representing three classes of differential equations: nonlinear, control affine, and linear, what conclusions can

we come to regarding their solutions? Is it always possible to solve a differential equation? If a solution is guaranteed, is it unique?
In this section, we'll explore some properties of *initial value problems*, which seek to find a solution $x(t)$ that solves a differential equation:

$$\dot{x} = f(x, u, t), \; x(t_0) = x_0 \tag{1.39}$$

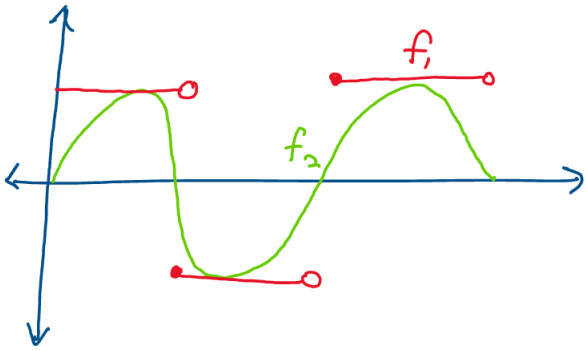
Subject to the initial condition $x(t_0) = x_0$.
To simplify our analysis, we'll primarily discuss the questions of existence and uniqueness for differential equations with no input ($u = 0$). Note that these results can be extended to systems with inputs without too much further effort.

Existence and Uniqueness of Solutions

Let's begin by thinking about the question of existence. How do we know a solution to a differential equation will exist? Although this might seem like a strange question at first, let's take a moment to think about why it might be relevant. Consider the time variant nonlinear system:

$$\dot{x} = f(x, t) \tag{1.40}$$

If $f(x, t)$ can be *any* nonlinear function of x, t, the possibilities for f are *endless*! f could be a sine function, a square wave function, a step function - the possibilities are virtually endless!



Above: We can choose anything for f in the general case.

For all of these choices of f, is it realistic to expect there to be a function $x(t)$ for which $\dot{x} = f(x, t)$ and $x(0) = x_0$? As it turns out, not quite! To understand the conditions required of f, we'll undertake a short discussion of continuity.
In mathematics, there are several formal definitions of what it means for a function to be continuous, with some definitions having stricter demands than others. Let's begin by discussing a strong type of continuity known as **Lipschitz continuity**.

Definition 1 ***Lipschitz Continuity***
A function $f(x) : \mathbb{R}^n \to \mathbb{R}^n$ is said to be globally Lipschitz continuous if there exists a finite scalar constant $L \in \mathbb{R}$ such that for all $x, y \in \mathbb{R}^n$:

$$||f(x) - f(y)|| \leq L||x - y|| \tag{1.41}$$

Let's think for a moment about what this definition of continuity states. Rearranging the definition of Lipschitz continuity, we require that for all $x, y \in \mathbb{R}^n$, there exists a constant L such that:

$$||f(x) - f(y)|| \leq L||x - y|| \tag{1.42}$$

$$\frac{||f(x) - f(y)||}{||x - y||} \leq L \tag{1.43}$$

For a simple interpretation of this inequality, we can think of the single variable case, where $n = 1$. This inequality then becomes:

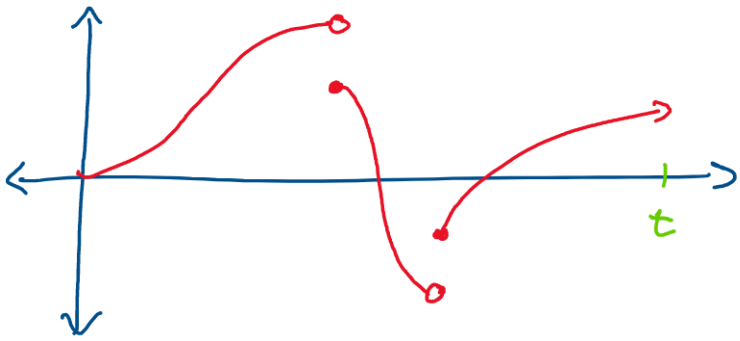
$$\frac{|f(x) - f(y)|}{|x - y|} \leq L \tag{1.44}$$

If we take the limit of both sides of this expression as $y \to x$, we observe that the Lipschitz condition bounds the derivative of the function by the Lipschitz constant!

$$\lim_{y \to x} \frac{|f(x) - f(y)|}{|x - y|} \leq \lim_{y \to x} L \tag{1.45}$$

$$\frac{df}{dx} \leq L \tag{1.46}$$

This means that a function such as a square wave, which has sudden jumps in its value across its domain, is *not* Lipschitz continuous.
Before we relate this material back to the study of differential equations, let's discuss a much less restrictive form of continuity, **piecewise continuity**. Although we won't state a formal mathematical definition, you can think of a piecewise continuous function as a bounded function with at most a finite number of breaks. For example, consider the function:



Above: A piecewise continuous function

Since this function has a finite number of breaks on the interval $[0, t]$ and remains bounded (doesn't "blow up" to infinity), it is said to be piecewise continuous on the domain $[0, t]$.
With these definitions in mind, we are now ready to understand the conditions for existence and uniqueness of solutions to differential equations.

Theorem 1 ***Global Existence and Uniqueness Theorem***
An initial value problem $\dot{x} = f(x,t)$, $x(t_0) = x_0$ has a unique solution if $f(x,t)$ is piecewise continuous with respect to t and for all $T \in [t_0, \infty)$, there exist finite constants $L_1, L_2 \in \mathbb{R}$ such that for all $t \in [0, T]$:

$$||f(x,t) - f(y,t)|| \leq L_1 ||x - y||, \text{ for all } x, y \in \mathbb{R}^n \tag{1.47}$$

$$||f(x_0,t)|| \leq L_2 \tag{1.48}$$

Let's break down the different parts of this theorem. First, we need f to be piecewise continuous with respect to t - as we vary t, there should only be a finite number of jumps in the value of f. Secondly, we require f to be globally Lipschitz continuous across all possible time intervals from 0 to ∞. Thirdly, we require the value of $f(x_0, t)$ to be bounded for all time. If these conditions are satisfied, the initial value problem is guaranteed to have a unique solution!
Let's try applying this theorem to show that a linear time invariant system will have a unique solution. Consider the linear system with constant A with all finite entries and a finite initial condition:

$$\dot{x} = Ax, \; x \in \mathbb{R}^n, \; x(t_0) = x_0 \tag{1.49}$$

Let's start out by identifying the function f! In this case, the function $f : \mathbb{R}^n \to \mathbb{R}^n$ is as follows:

$$f(x) = Ax \tag{1.50}$$

Since $f(x) = Ax$ is a constant function with respect to time, this means that it must be piecewise continuous with respect to time. This checks off the first required condition!
Next, let's show that f is globally Lipschitz continuous. Looking at the definition of Lipschitz continuity, we start by analyzing $||Ax - Ay||$, where $x, y \in \mathbb{R}^n$ are arbitrary vectors. We know:

$$||Ax - Ay|| = ||A(x - y)|| \tag{1.51}$$

Where do we go from here? We'd like to find some way of bringing A out of the norm, $||\cdot||$, and forming an inequality. If we can do this, we'll have demonstrated that $||Ax - Ay|| \leq L||x - y||$ for some finite L!
How can we extract A? One way of doing this is to use something known as a *matrix norm*. Just like we can assign a norm to a vector to measure its size, we can do the same thing for a matrix! There are many ways of doing this. The matrix 2-norm is defined as follows:

$$||A||_2 = \sup_{x \neq 0 \in \mathbb{R}^n} \frac{||Ax||}{||x||} = \sigma_{max} \tag{1.52}$$

Where $\sup_{x \neq 0 \in \mathbb{R}^n}$ refers to the *supremum* of $||Ax||/||x||$, the smallest possible upper bound of $||Ax||/||x||$ as we change the value of x. Miraculously, it can be shown that this norm is equal to the maximum singular value of of A, σ_{max}!
By definition of the supremum, σ_{max} is an upper bound for $||Ax||/||x||$. This means that for all $x \in \mathbb{R}^n$, we can form the inequality:

$$\frac{||Ax||}{||x||} \leq \sigma_{max} \tag{1.53}$$

$$||Ax|| \leq \sigma_{max}||x|| \tag{1.54}$$

Let's apply this back to our discussion of Lipschitz continuity! Recall:

$$||Ax - Ay|| = ||A(x - y)|| \tag{1.55}$$

Applying our result from the 2-norm of A, we reach the inequality:

$$||Ax - Ay|| \leq \sigma_{max}||x - y|| \tag{1.56}$$

Thus, as long as the entries of A are all finite, $f(x) = Ax$ is a Lipschitz continuous function! This checks off the second requirement for global existence and uniqueness.
Finally, we must check the third result: that there exists an L_2 such that $||f(x_0, t)|| \leq L_2$ for all t. Since $f(x) = Ax$ is not a function of t, this simplifies the requirement to:

$$||f(x_0)|| = ||Ax_0|| \leq L_2 \tag{1.57}$$

For some L_2. Since $||Ax_0||$ is a constant, as long as A and x_0 have all finite entries, this expression is bounded. Thus, the third condition is checked off.

Since $\dot{x} = Ax, x(t_0) = x_0$ satisfy all of the required conditions, we conclude that linear time invariant systems *must* have a unique solution $x(t)$ for all finite initial conditions x_0.
Note that we can also extend this result with minimal further conditions to the case where the system has a nonzero input, u, and the matrix A is time-varying.

Solving Linear Differential Equations

Now that we've shown that a unique solution to a linear differential equation must exist for all well-behaved initial conditions, let's try finding what the solution actually is!
We'll start by dealing with the case where the system has no input. Consider the linear, time invariant system:

$$\dot{x} = Ax,\ x \in \mathbb{R}^n,\ x(t_0) = x_0 \tag{1.58}$$

How would we go about finding a solution to this differential equation? Let's begin by reviewing the *scalar* case, where $n = 1$. The scalar case of this differential equation would be:

$$\dot{x} = \frac{dx}{dt} = ax,\ x(t_0) = x_0 \tag{1.59}$$

To solve this equation, we can use a technique known as *separation of variables*, where we bring all of the quantities involving x to one side of the equation and all quantities involving t to the other side. Let's begin this process by expanding the derivative $\frac{dx}{dt}$:

$$\frac{dx}{dt} = ax \tag{1.60}$$

$$\frac{dx}{x} = adt \tag{1.61}$$

Now, we may integrate both sides. On the left hand side, we will integrate from x_0 to x, and on the right hand side, from t_0 to t. Note that when integrating, we use "dummy variables" χ and τ in the place of x and t for the sake of proper mathematical convention.

$$\int_{x_0}^{x} \frac{1}{\chi} d\chi = \int_{t_0}^{t} a d\tau \tag{1.62}$$

$$[ln|\chi|]_{x_0}^{x} = [a\tau]_{t_0}^{t} \tag{1.63}$$

$$ln\frac{|x|}{|x_0|} = a(t - t_0) \tag{1.64}$$

Where we applied a logarithm rule to turn $ln(a) - ln(b)$ into $ln(a/b)$. Assuming that $x(t)$ and x_0 have the same sign, we drop the absolute values and solve for x by exponentiating both sides.

$$\frac{x}{x_0} = e^{a(t-t_0)} \tag{1.65}$$

$$x(t) = e^{a(t-t_0)} x_0 \tag{1.66}$$

Thus, we have found the solution to our differential equation! Let's turn back to the case where $x \in \mathbb{R}^n$, and use the scalar case to guide us to a vector solution. We'd now like to solve the initial value problem:

$$\dot{x} = Ax,\ x \in \mathbb{R}^n,\ x(t_0) = x_0 \tag{1.67}$$

Since we found that the solution to the scalar case was $e^{a(t-t_0)}x_0$, we hypothesize that the solution to the vector case will be:

$$x(t) = e^{A(t-t_0)}x_0 \tag{1.68}$$

Where $e^{A(t-t_0)}$ is the **matrix exponential** of $A(t-t_0)$, defined according to the Taylor series of e^x.

$$e^{At} = I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + ... = \sum_{k=1}^{\infty} \frac{(At)^k}{k!} \in \mathbb{R}^{n\times n} \tag{1.69}$$

We can prove that for all finite A, this series will actually converge to an $n \times n$ matrix! Using this definition, let's show that our hypothesized solution satisfies the differential equation $\dot{x} = Ax$ and the initial condition $x(t_0) = x_0$.
Let's first check that this solution satisfies the differential equation. We can start by taking the time derivative of $x(t)$:

$$\dot{x}(t) = \frac{d}{dt}\Big(e^{A(t-t_0)}\Big)x_0 \tag{1.70}$$

We may use the series definition of the matrix exponential to find the derivative of $e^{A(t-t_0)}$.

$$\frac{d}{dt}\Big(e^{A(t-t_0)}\Big) = \frac{d}{dt}\Big[I + A(t-t_0) + \frac{(A(t-t_0))^2}{2!} + \frac{(A(t-t_0))^3}{3!} + ...\Big] \tag{1.71}$$

$$= 0 + A + A^2(t-t_0) + \frac{A^3(t-t_0)^2}{2!} + ... \tag{1.72}$$

$$= A\Big[I + A(t-t_0) + \frac{(A(t-t_0))^2}{2!} + ...\Big] \tag{1.73}$$

$$= Ae^{A(t-t_0)} \tag{1.74}$$

Let's substitute this into our expression for the time derivative of x and see what we get!

$$\dot{x} = \frac{d}{dt}\Big(e^{A(t-t_0)}\Big)x_0 \tag{1.75}$$

$$\dot{x} = Ae^{A(t-t_0)}x_0 \tag{1.76}$$

Now, we notice that $e^{A(t-t_0)}x_0$ equal to our proposed solution, $x(t)$! Thus:

$$\dot{x} = Ax \tag{1.77}$$

This solution therefore satisfies our differential equation. Using the definition of the matrix exponential, we can also check that this solution satisfies the initial condition. Thus, the matrix exponential provides us with the unique solution to a linear, time invariant differential equation with no input for an arbitrary initial condition.
Is there a similarly general solution to the case where we *do* apply an input to the system? Although we won't prove it here, we can show that the solution to the initial value problem:

$$\dot{x} = Ax + Bu,\ x \in \mathbb{R}^n, u \in \mathbb{R}^m,\ x(t_0) = x_0 \tag{1.78}$$

Is given by the following expression:

$$x(t) = e^{A(t-t_0)}x_0 + \int_{t_0}^{t} e^{A(t-\tau)}Bu(\tau)d\tau \tag{1.79}$$

Notice that if $u(t) = 0$ for all time, this solution simplifies to the zero-input case discussed above!

Evaluating the Matrix Exponential

So far, we've shown that the solution to a variety of linear systems relies on the use of the matrix exponential. How can we actually compute what the value of the matrix exponential is?
Before we discuss techniques for computing the exponential, it's important to remember - the matrix exponential is *not* simply the exponential of each entry in the matrix! That is, if we have a matrix:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.80}$$

Its matrix exponential is *not* generally computed by taking the exponential of each a_{ij}.

$$e^{At} \neq \begin{bmatrix} e^{a_{11}t} & \dots & e^{a_{1n}t} \\ \vdots & \ddots & \vdots \\ e^{a_{n1}t} & \dots & e^{a_{nn}t} \end{bmatrix} \text{ (For general } A \in \mathbb{R}^{n \times n}) \tag{1.81}$$

How, then, can we compute its value? The matrix exponential is simplest to compute for a diagonal matrix. Suppose we have a diagonal matrix A, defined:

$$A = \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.82}$$

From linear algebra, we know that we can compute the p^{th} power of a diagonal matrix by raising each diagonal entry to the p^{th} power:

$$A^p = \begin{bmatrix} \lambda_1^p & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n^p \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.83}$$

Let's use this fact to compute the matrix exponential of a diagonal A. Recall:

$$e^{At} = I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + ... \tag{1.84}$$

$$= \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} + \begin{bmatrix} \lambda_1 t & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n t \end{bmatrix} + \begin{bmatrix} \frac{\lambda_1^2 t^2}{2!} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \frac{\lambda_n^2 t^2}{2!} \end{bmatrix} + ... \tag{1.85}$$

$$= \begin{bmatrix} 1 + \lambda_1 t + \frac{\lambda_1^2 t^2}{2!} + & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 + \lambda_n t + \frac{\lambda_n^2 t^2}{2!} + ... \end{bmatrix} \tag{1.86}$$

Now, we recognize each diagonal entry as the scalar Taylor series of $e^{\lambda_i t}$. Thus, for diagonal A, we conclude:

$$e^{At} = \begin{bmatrix} e^{\lambda_1 t} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & e^{\lambda_n t} \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.87}$$

Using this simple diagonal structure, we may devise a method to compute the matrix exponential of *any* diagonalizable matrix!
Suppose we have a diagonalizable matrix $A \in \mathbb{R}^{n \times n}$. Since A is diagonalizable, we know there exists a transformation matrix $T \in \mathbb{R}^{n \times n}$ such that:

$$D = TAT^{-1} \tag{1.88}$$

$$A = T^{-1}DT \tag{1.89}$$

Where D is a diagonal matrix with all of the eigenvalues of A along its diagonal!

$$D = \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.90}$$

Let's see if we can compute the matrix exponential of A using what we now know about diagonal matrices.
Using the transformation defined above, let's take the matrix exponential of

$At = T^{-1}DTt$ and see if we can extract e^{Dt}, which we know how to compute.

$$e^{At} = e^{T^{-1}DTt} \tag{1.91}$$

$$e^{At} = I + T^{-1}DTt + \frac{(T^{-1}DTt)^2}{2!} + \frac{(T^{-1}DTt)^3}{3!} + ... \tag{1.92}$$

$$e^{At} = I + T^{-1}DTt + \frac{T^{-1}DTT^{-1}DTt^2}{2!} + \frac{(T^{-1}DTt)^2T^{-1}DTt}{3!} + ... \tag{1.93}$$

$$e^{At} = I + T^{-1}(Dt)T + \frac{T^{-1}(Dt)^2T}{2!} + \frac{T^{-1}(Dt)^3T}{3!} + ... \tag{1.94}$$

$$e^{At} = T^{-1}\Big[I + Dt + \frac{(Dt)^2}{2!} + \frac{(Dt)^3}{3!} + ...\Big]T \tag{1.95}$$

$$e^{At} = T^{-1}e^{Dt}T \tag{1.96}$$

Therefore, to calculate e^{At} for any diagonalizable A, all we need to do is calculate the matrix exponential of the associated diagonal matrix, e^{Dt}, and transform it *back* to e^{At} using $T^{-1}e^{Dt}T$. This enables us to compute the closed form of the matrix exponential for a wide variety of matrices.
Note that if A is *not* a diagonalizable matrix, it may be transformed into another form called the **Jordan canonical form** (JCF) for easy computation of the matrix exponential. Interested readers are encouraged to consult *Linear Algebra* by Friedberg, Insel, and Spence for an in-depth treatment of the Jordan canonical form.

1.1.3 Equilibrium Points

We now have a significant body of language which we may use to understand and interpret dynamical systems. Let's begin the process of analyzing these systems, and examine their key points in closer detail.
The first concept we'll study is that of an **equilibrium point**.

Definition 2 ***Equilibrium Point***
An equilibrium point of a system $\dot{x} = f(x, u)$ is a pair (x_e, u_e) such that:

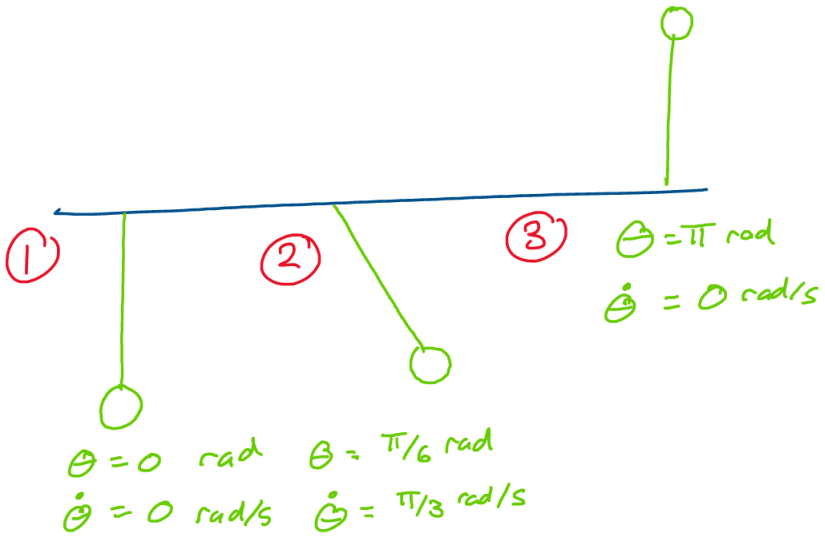
$$0 = f(x_e, u_e) \tag{1.97}$$

Let's break this definition down and interpret the name "equilibrium point" physically. If we are at an equilibrium point (x_e, u_e), this means that the derivative of the state vector of the system is zero:

$$\dot{x} = 0 = f(x_e, u_e) \tag{1.98}$$

This means that at this point, the evolution of the system is entirely *frozen* - none of the state variables are changing with respect to time!

What might an equilibrium point look like in a physical system? Let's consider the example of a simple pendulum to gain some intuition. We'll then proceed to verify our results mathematically.



Above: Three different positions and velocities of a simple pendulum

Let's consider the first position-velocity pair, where the pendulum sits in a vertically downward position with zero angular velocity. In this configuration, will the state vector be changing with respect to time?

Intuitively, we know that if we apply no forces to a pendulum in its downwards position, it will stay there - neither the position nor velocity of the pendulum should change! Thus, we hypothesize that this first configuration is an equilibrium point of the system.

What about the second configuration? This configuration captures the pendulum mid-swing - will this be an equilibrium point? Since the pendulum is actively swinging in this configuration, we know that the system will not remain in that configuration as time goes on. The pendulum will continue to swing up and down. This second configuration is therefore not an equilibrium point.

What about the third configuration? In this configuration, the pendulum is perfectly balanced at 180 degrees from the downwards position. Assuming that there are no small disturbances, the pendulum should remain perfectly balanced in this configuration. Thus, we hypothesize that the third configuration is also an equilibrium point.

Notice how although the first and third configurations are both equilibrium points, the behavior of the pendulum at and around those points is quite different! If we tried to balance the pendulum by hand in its lower position, we would find it to be quite easy.

If we were to try and balance the pendulum in its upper configuration, however, it would be significantly more challenging, as it would have a tendency to fall

right down! We'll soon learn how to formalize these qualitative differences as we shift our discussion to the topic of stability.
Let's mathematically verify our hypotheses about which configurations are equilibrium points and which are not.The dynamics of a simple frictionless pendulum with length l, mass m, and no external forces are expressed:

$$ml^2\ddot{\theta} = -mgl\sin\theta \tag{1.99}$$

Let's convert this equation into phase variable form and solve for its equilibrium points. Choosing $\theta, \dot{\theta}$ as state variables, we may rewrite this system as:

$$\dot{q} = \begin{bmatrix}\dot{\theta}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix}\dot{\theta}\\ -\frac{g}{l}\sin\theta\end{bmatrix} = f(q) \tag{1.100}$$

To solve for the equilibrium points, we set $f(q) = 0$.

$$\begin{bmatrix}0\\0\end{bmatrix} = \begin{bmatrix}\dot{\theta}\\ -\frac{g}{l}\sin\theta\end{bmatrix} \tag{1.101}$$

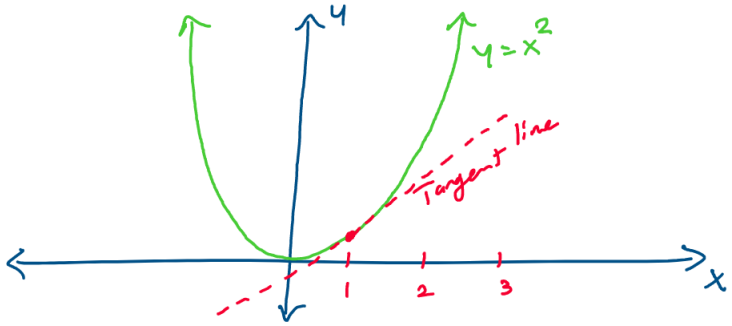
$$0 = \dot{\theta} \tag{1.102}$$

$$0 = \sin\theta \tag{1.103}$$

Thus, all points of the form $(n\pi, 0) = (\theta, \dot{\theta})$, where $n \in \mathbb{Z}$ is an integer, must be equilibrium points of the system!
Does this match up with our earlier hypotheses? Since even values of n correspond to the downwards position of the pendulum and odd values of n correspond to the upwards position of the pendulum, we conclude that our earlier analyses were correct - both the upwards and downwards configurations are equilibrium points.

1.1.4 Linearization

Thus far in our treatment of dynamical systems, we've considered linear and nonlinear systems largely as separate groups of systems. As we've seen thus far, linear systems have simple and interpretable solutions compared to nonlinear systems, and allow us to apply the often-elegant tools of linear algebra in our analyses.
Is there some way we can meaningfully *relate* a nonlinear system to a linear system? If we can achieve this, we can apply the tools of linear analysis to study nonlinear systems. To answer this question, let's take a moment to turn our attention away from the study of dynamical systems and towards the study of calculus.
One of the most fundamental concepts in calculus is that of the derivative. We know that using the derivative of a function, we can find the line tangent to a function at any point where its derivative is defined. For instance, consider the function $y = x^2$, which has been sketched below.



Above: The function $y = x^2$. How do we find the equation of its tangent line?

Suppose we wanted to find the equation for the line tangent to the curve at $x = 1$. Firstly, we could find the slope of the tangent line by evaluating the derivative of the function at $x = 1$.

$$\frac{dy}{dx}\Big|_{x=1} = 2 \tag{1.104}$$

Note that the notation $dy/dx|_{x=1}$ means evaluate the value of dy/dx at $x = 1$. After we have the slope, to find the equation of the tangent line, all we need are the coordinates of a point on the line. In this case, we can pick $(1, 1)$, the point of intersection of the tangent line with the function. The equation of the tangent line is then:

$$y = 2(x - 1) + 1 \tag{1.105}$$

We can generalize this equation to find the tangent line at *any* point where a function is differentiable. The line tangent to a function $f(x)$ at a point x_t is given by:

$$y = \left(\frac{df}{dx}\Big|_{x=x_t}\right)(x - x_t) + f(x_t) \tag{1.106}$$

Let's take a moment to think critically about the applications of this equation. When we found the tangent line of the *nonlinear* function $y = x^2$ at the point $x = 1$, we found a *linear* function that, in a small region around $x = 1$, looked similar to the original, nonlinear function. This is something we can observe if we zoom in close to the point $x = 1$:

Above: In a small region around $x = 1$, the tangent line and the nonlinear function appear to be similar.

With this similarity in mind, could we use the tangent line as a local linear approximation of the nonlinear function?
Let's perform an informal mathematical study of this question, and see how the difference between the nonlinear function and tangent line change as we move further away from $x = 1$. Let's defined the error between the nonlinear function and tangent line to be:

$$e(x) = y_{nonlinear} - y_{tangent} \tag{1.107}$$
$$e(x) = x^2 - 2(x-1) - 1 \tag{1.108}$$
$$e(x) = x^2 - 2x + 1 \tag{1.109}$$
$$e(x) = (x-1)^2 \tag{1.110}$$

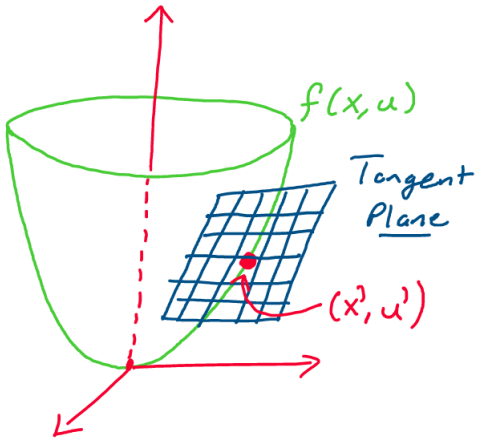
Thus, as long as we don't move too far away from the tangent point, $x = 1$, the error between the actual function and approximation will remain small! If we move away from the tangent point, our error will grow at an unbounded, quadratic rate. Thus, we can use a tangent line for a good *local* approximation of the nonlinear function!
Let's discuss how we can apply this concept to locally approximate nonlinear systems as linear ones! We'd like to approximate a nonlinear system of the form:

$$\dot{x} = f(x,u),\ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{1.111}$$

As a linear system:

$$\dot{z} = Az + Bv,\ z \in \mathbb{R}^n, v \in \mathbb{R}^m \tag{1.112}$$

For *some* choice of A, B, z, v. How can we accomplish this? Since our functions are now multivariable, instead of finding a tangent line to approximate a curve, we now look for a tangent *plane* that approximates a surface!



Above: We wish to find the equation of the plane tangent to a surface.

Let's begin by finding the equation of the plane tangent to $f(x,u)$ at the point (x', u'). To find the equation of the tangent plane, we follow the same procedure

as for finding a tangent line! The tangent plane is defined:

$$\frac{\partial f}{\partial x}\Big|_{(x,u)=(x',u')}(x-x') + \frac{\partial f}{\partial u}\Big|_{(x,u)=(x',u')}(u-u') + f(x',u') \tag{1.113}$$

Where $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial u}$, known as the **Jacobians** of f with respect to x and u, are defined:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{1.114}$$

$$\frac{\partial f}{\partial u} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_m} \end{bmatrix} \in \mathbb{R}^{n\times m} \tag{1.115}$$

Notice that the definition of the tangent plane follows the same format as that of a tangent line! We first take the derivative of the nonlinear function with respect to x, evaluate it at the tangent point, and multiply it by the difference between x and the point of approximation. Since this is a multivariable function, we repeat this procedure for u. Following this, we add on the value of the function at the point (x', u').

Now that we have the equation of the tangent plane at our point of approximation, we notice a few problems! This equation looks *nothing* like our desired format for a linear system:

$$\dot{z} = Az + Bv \tag{1.116}$$

The first step we'll take towards transforming our tangent plane equation to the correct, linear system form is defining a change of variables. We want a change of variables such that we have one matrix multiplied by a vector z and another matrix multiplied by a vector v.

Looking at our formula for the tangent plane, we notice that we have one Jacobian matrix multiplied by $x - x'$ and another Jacobian matrix multiplied by $u - u'$. Based on this observation, we define z and v as follows:

$$z = x - x' \tag{1.117}$$

$$v = u - u' \tag{1.118}$$

Rewriting our tangent plane equation in terms of z and v, we have:

$$\frac{\partial f}{\partial x}\Big|_{(x,u)=(x',u')} z + \frac{\partial f}{\partial u}\Big|_{(x,u)=(x',u')} v + f(x',u') \tag{1.119}$$

This is much closer to the format we're looking for! We have one remaining term we'd like to get rid of: $f(x', u')$. If we can somehow ensure $f(x', u') = 0$, we'll have found our linear approximation of the nonlinear system!

We know that $f(x', u') = 0$ if (x', u') is an equilibrium point of the system $\dot{x} = f(x, u)$. Thus, to ensure our linear approximation is valid, we require that the point at which we take our linear approximation is an equilibrium point, $(x', u') = (x_e, u_e)$, of the nonlinear system. Applying the assumption that the point of approximation is an equilibrium point, we finally arrive at the following equation for the tangent plane:

$$\frac{\partial f}{\partial x}\Big|_{(x,u)=(x',u')} z + \frac{\partial f}{\partial u}\Big|_{(x,u)=(x',u')} v \tag{1.120}$$

Therefore, if we define A, B, z, v as follows:

$$A = \frac{\partial f}{\partial x}\Big|_{(x,u)=(x',u')} \tag{1.121}$$

$$B = \frac{\partial f}{\partial u}\Big|_{(x,u)=(x',u')} \tag{1.122}$$

$$z = x - x' = x - x_e \tag{1.123}$$

$$v = u - u' = u - u_e \tag{1.124}$$

We may approximate the nonlinear system $\dot{x} = f(x, u)$ by its tangent plane at its equilibrium points as:

$$\dot{z} = Az + Bv \tag{1.125}$$

This linear system is known as the **Jacobian linearization** of the nonlinear system, due to its use of the Jacobians of $f(x, u)$ in defining A and B.
Let's summarize the process of taking a Jacobian linearization of a nonlinear system with a step by step procedure.

Definition 3 ***Jacobian Linearization***
To find the Jacobian linearization of a nonlinear system, $\dot{x} = f(x, u)$, follow the procedure:

1. *Ensure that the point at which you are taking your linearization is an equilibrium point (x_e, u_e), where $f(x_e, u_e) = 0$.*
2. *Find the Jacobians of f with respect to x and u, and evaluate them at the point $x = x_e, u = u_e$. Define these to be the A and B matrices of your approximated system.*

$$A = \frac{\partial f}{\partial x}\Big|_{(x,u)=(x_e,u_e)} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.126}$$

$$B = \frac{\partial f}{\partial u}\Big|_{(x,u)=(x_e,u_e)} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_m} \end{bmatrix} \in \mathbb{R}^{n \times m} \tag{1.127}$$

3. Define a change of variables:

$$z = x - x_e \tag{1.128}$$
$$u = u - u_e \tag{1.129}$$

4. Write the approximate linear system in terms of A, B, z, v:

$$\dot{z} = Az + Bv \tag{1.130}$$

Generally, this approximation is only valid within a small region surrounding the equilibrium point (x_e, u_e).

As we move forward into our study of stability, we'll find the Jacobian linearization to be a powerful tool that enables us to make nontrivial conclusions about the behavior of the nonlinear system using linear algebra.

1.1.5 Discrete Time Systems

Thus far, our discussion of dynamical systems has focused strictly on the study of continuous time systems. Continuous time systems are systems whose states evolve smoothly as time passes. These are the types of systems we deal with in the real world.

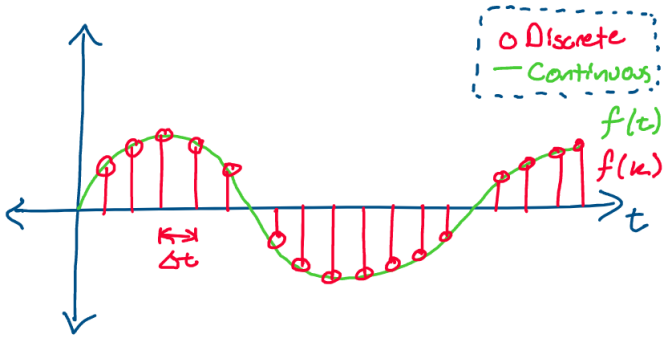
As an apple falls from a tree or a drone flies across a field, its state will change continuously as every second, microsecond, or any other increment of time passes by. These continuous time systems are *all* governed by some form of differential equation. Are there any other types of systems out there?

Discrete time systems are another important class of dynamical system. Instead of caring about the state of the system at *every* instant in time, a discrete time system jumps from one state to the next at *fixed time intervals*.

Our digital computers are one example of a discrete time system - they perform computations and interact with the physical world in small discrete intervals.

The fact that the *physical* systems we try to control are continuous, but the computers we might use to control them are discrete brings up an important point in the study of dynamical systems. Since our computers operate in discrete time, and perceive the physical world through sampling from sensors at set time intervals, it's important to have an understanding of how we can model discrete time systems.

What characterizes a discrete time signal? Consider the following graph, where a discrete time sine wave, plotted in red, and a continuous time sine wave, plotted in green, have been overlaid. Note that a discrete time signal is typically drawn with a circle at the value of the signal and a line extending to the time axis.



Above: A digital signal (in red) and a continuous signal (in green).

The first, and perhaps most important quality of a discrete time signal is that it's only defined at *discrete time intervals* - we only care about the value of a discrete signal at certain instants in time. The **sampling time**, Δt, is the time between these intervals.

What consequences does this characteristic have for our description of discrete time systems? If we only care about the value of a signal at increments of Δt, we can track the total time that has passed by taking integer multiples of Δt:

$$t = k\Delta t,\ k = 0, 1, 2, ... \in \mathbb{Z}_0^+ \tag{1.131}$$

Where $\mathbb{Z}_0^+$ is the set of positive integers including zero. Thus, if the sampling time Δt is fixed, all we need to describe t in a discrete time system is a single integer, k, the number of Δt intervals that have passed.

Because all we need to describe time is k, discrete time signals are typically written as functions of the form:

$$x(k),\ k = 0, 1, 2, ... \in \mathbb{Z}_0^+ \tag{1.132}$$

Let's consider some examples of discrete time signals described in this manner. A discrete time sine wave with amplitude a, frequency ω, and sampling period Δt could be written:

$$x(k) = a\sin(\omega k\Delta t) \tag{1.133}$$

Using functions such as these, we can construct discrete time systems that are every bit as rich and interesting as continuous time systems.

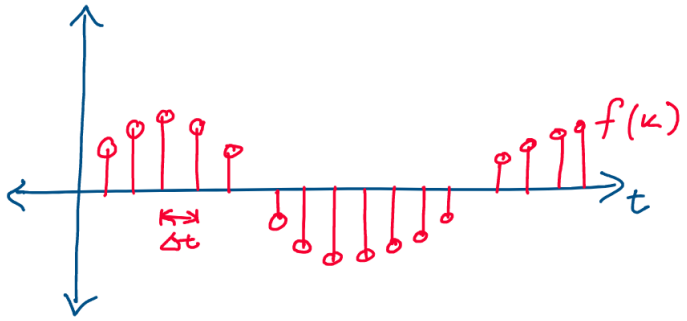
Now that we've come up with a method of describing discrete time signals, let's discuss how we can formulate discrete time dynamical systems. Recall that for continuous time systems, we use models of the form:

$$\dot{x} = f(x, u) \tag{1.134}$$

Does this description still make sense for a discrete time system? For this model to be applicable to a discrete time system, the time derivative of the state vector must be well-defined! This means that the limit:

$$\frac{dx}{dt} = \lim_{\delta t \to 0} \frac{x(t + \delta t) - x(t)}{\delta t} \tag{1.135}$$

Must exist. Does this limit make sense for a discrete time signal $x(k)$? Consider the following discrete time signal:



Above: does it make sense to take the derivative of this discrete signal?

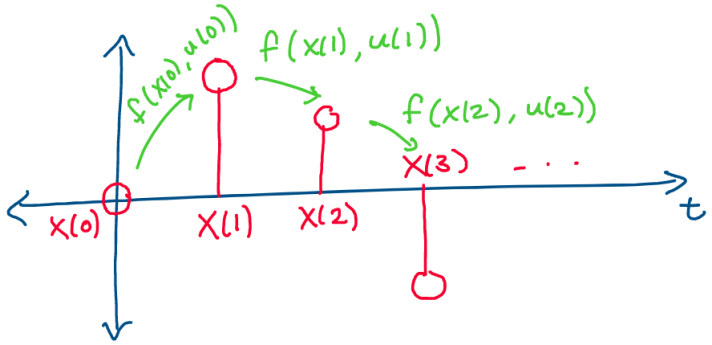
Since this signal is only defined at individual instants, it is not differentiable with respect to time at *any* t! Thus, we have no way of taking the limit as $\delta t \to 0$. This means that we *cannot* use the description $\dot{x} = f(x, u)$ to describe a discrete time dynamical system.

How, then, can we describe the evolution of such a system? Instead of using the derivative of the state vector to describe the evolution of a system, discrete time systems use a function f that maps from the state and input vectors at time step k to the state vector at time step $k + 1$.

Mathematically, the conventional representation of a discrete time system with a state x and an input u is written:

$$x(k+1) = f(x(k), u(k)),\ x \in \mathbb{R}^n,\ u \in \mathbb{R}^m,\ k \in \mathbb{Z}_0^+ \tag{1.136}$$

As you can see in the equation above, instead of taking in an input and state and returning a state derivative, the function f takes in an input and a state at step k and returns the state at step $k + 1$.



Above: $f(x(k), u(k))$ maps to the next state vector, $x(k + 1)$.

Just like with continuous time systems, there are several important subclasses of discrete time systems, all with the same definitions as their continuous time

counterparts. There are control affine discrete time systems, which have the form:

$$x(k+1) = f(x(k)) + g(x(k))u(k) \tag{1.137}$$

And linear discrete time systems, which have the form:

$$x(k+1) = Ax(k) + Bu(k) \tag{1.138}$$

Where $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$ are matrices. Note that in the most general case, the mapping from state and input to next state may also be an explicit function of the time step, k.

Discretization

Since we often use digital computers to design our controllers, in many control design problems, it proves useful to approximate a continuous time system by a discrete time system! This is particularly common in model predictive control, which formulates a control and path planning problem as a large optimization problem. We'll discuss this strategy in detail in the coming sections!
The process of approximating a continuous time system by a discrete time system is known as **discretization**. How can we perform discretization in a meaningful way that *preserves* some behavior of our original system?
Let's begin our analysis of this problem with the general nonlinear system:

$$\dot{x} = f(x, u) \tag{1.139}$$

Where we assume f is a smooth mapping. Let's see if we can extract a discrete time model from this continuous system. To begin, recall the limit definition of a derivative:

$$\dot{x} = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - x(t)}{\Delta t} \tag{1.140}$$

Instead of taking the limit of this expression as $\Delta t \to 0$, let's try *approximating* the value of the derivative by fixing Δt to be some small constant. Then, for small Δt:

$$\dot{x} \approx \frac{x(t + \Delta t) - x(t)}{\Delta t} \tag{1.141}$$

Since $\dot{x} = f(x, u)$, this tells us:

$$\frac{x(t + \Delta t) - x(t)}{\Delta t} \approx f(x, u) \tag{1.142}$$

$$x(t + \Delta t) - x(t) \approx f(x, u)\Delta t \tag{1.143}$$

$$x(t + \Delta t) \approx f(x, u)\Delta t + x(t) \tag{1.144}$$

This format, where we have $x(t + \Delta t)$ on the left hand side, and $x(t)$ on the right hand side, looks similar to the format of a discrete time system! Let's

define the sampling time of our discrete time system to be Δt, the small value that we chose when approximating the derivative of the state vector.
Using this Δt, we can express the total time passed at each sampling interval as $t = k\Delta t$, where k is an integer. Let's plug this into the expression above and see what we get!

$$x(k\Delta t + \Delta t) \approx f(x(k\Delta t), u(k\Delta t))\Delta t + x(k\Delta t) \tag{1.145}$$
$$x((k+1)\Delta t) \approx f(x(k\Delta t), u(k\Delta t))\Delta t + x(k\Delta t) \tag{1.146}$$

Thus, we can now *entirely* identify the dynamics of the system with a single integer k! To convert into the correct notation, we define a discrete time estimate of the state vector, $\hat{x}(k) = x(k\Delta t)$ and an input vector $\hat{u}(k) = u(k\Delta t)$. Notice that we *only* use the values of the state and input vector *exactly* at $t = k\Delta t$. We lose any information about how $x(t), u(t)$ vary during each sampling period. This enables us to rewrite our approximation as:

$$\hat{x}(k+1) = f(\hat{x}(k), \hat{u}(k))\Delta t + \hat{x}(k) \tag{1.147}$$

We have successfully approximated our continuous time system with a discrete time system! This type of approximation is known as **Euler discretization**, and can yield fairly good results over short time periods for small Δt.
If we were to apply this approximation for long periods of time, however, note that the error between $x(t)$ and $\hat{x}(k)$ might accumulate at an unbounded rate. To slow the accumulation of error, other methods of discretization that use higher derivatives of the state vector, such as **Runge-Kutta discretization**, may be used.

1.2 Mathematical Preliminaries

Before we progress further with our analysis of dynamical systems, let's take a moment to develop some important concepts from real analysis, the field of mathematics concerned with rigorously proving results in calculus. These concepts will be particularly helpful in describing the stability of dynamical systems. Note that we'll develop this material primarily for the purpose of being precise in stating our theorems and definitions later down the line! Don't feel any pressure to be fully proficient in wielding these as mathematical tools in proofs. Interested readers are encouraged to refer to texts such as *Understanding Analysis* by Abbott and *Principles of Mathematical Analysis* by Rudin for a comprehensive treatment of the concepts we discuss here.

1.2.1 Quantifiers

When discussing advanced concepts, the formal language of mathematics can often get quite convoluted and cluttered! We often find ourselves writing proofs regarding existence and uniqueness and making conclusions about whether certain results hold for all values of a certain variable or only for certain values. When writing proofs, this means that we oftentimes have to write an enormous amount just to be precise in what we mean.

To ease this proof-writing clutter, mathematicians conventionally use a set of symbols known as **quantifiers** to assist in writing cleaner proofs and definitions. These symbols are also frequently used in papers in robotics! We'll occasionally make use of them in our discussion of stability to ensure our definitions remain concise.

The first quantifier we'll discuss is **for all**, which has the symbol $\forall$. Let's get some practice using this symbol in context. The line:

$$\forall t \geq t_0 \tag{1.148}$$

Reads "for all t greater than or equal to t_0." Another common use of this symbol is in describing sets of vectors. For example:

$$\forall x \in \mathbb{R}^n \tag{1.149}$$

Reads "for all x in $\mathbb{R}^n$." This is a useful quantifier when expressing results that hold globally for all possible values of a variable.

The second quantifier we'll make use of is **there exists**, which has the symbol $\exists$. If we write:

$$\exists\, x \in \mathbb{R}^n : f(x) = 0 \tag{1.150}$$

We are communicating that "there exists an x in $\mathbb{R}^n$ such that $f(x) = 0$."

Now that we have a symbol for existence, we can turn our attention to uniqueness. Recall that mathematically, something is said to be unique if it is the *only* object that possesses a certain property. If we have a variable that **exists** and is

unique, we use the quantifier $\exists!$, which is simply $\exists$ with an exclamation mark. In context:

$$\exists!\, x \in \mathbb{R}^n : f(x) = 0 \tag{1.151}$$

Reads "there exists a *unique* x in $\mathbb{R}^n$ such that $f(x) = 0$." This means that this x is the only one in $\mathbb{R}^n$ for which $f(x) = 0$.
Notice how using these symbols can save us a considerable amount of writing when describing properties of mathematical objects!

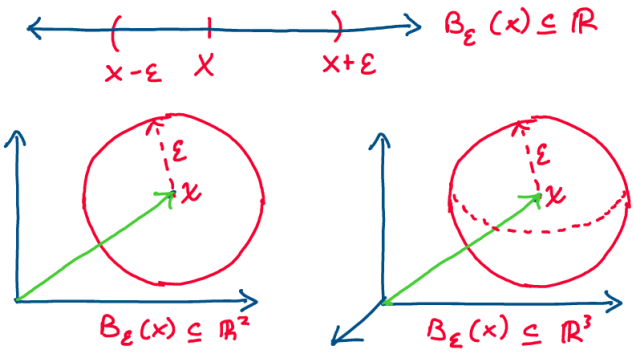
1.2.2 Neighborhoods

Now that we have a little bit more notation at our disposal, we can progress to a discussion of some fundamental concepts in real analysis. In the next section, when coming up with a definition of stability, we will rely on having some mathematical idea of *closeness* to an equilibrium point.
In analysis, we often formalize the concept of two mathematical objects being close using a **neighborhood**. Let's discuss neighborhoods in $\mathbb{R}^n$.
A neighborhood of a point p in $\mathbb{R}^n$ is a region of space that is within a certain radius of the point p. We can use neighborhoods to formally reason about the closeness of different points in space. For example, if a point q is contained in a neighborhood of a small radius of a point p, we can conclude that p and q are close together! Let's formally define this concept.

Definition 4 ***Neighborhood (Open Ball)***
A neighborhood of radius ε about a point $p \in \mathbb{R}^n$, denoted $B_\varepsilon(p)$, is defined:

$$B_\varepsilon(p) = \{x \in \mathbb{R}^n : \ ||x - p|| < \varepsilon\} \tag{1.152}$$

As we can see from the definition above, a neighborhood of a point is simply the set of all points within a sphere with its center at p and a radius of ε. In one, two, and three dimensions, we can therefore visualize neighborhoods as follows:



Above: An neighborhood in 1, 2, and 3 dimensions.

Because of their sphere-like shape, neighborhoods are also often referred to as an **open balls**. Note that the word *open* comes from the fact that we use a strict inequality, $||x - p|| < \varepsilon$, instead of $||x - p|| \leq \varepsilon$ in our definition. This simply means that we don't include the surface of the sphere in our set, which makes the shape "open."

1.2.3 Bounds

Oftentimes, when we have a set of different values, for example:

$$A = \{x^2 \mid 0 \leq x \leq 1\} \tag{1.153}$$

It's important to know if the set is *bounded* or not! This can help us reach important conclusions about the behavior of objects in the set. How can we formally define the boundedness of sets?
There are two types of bounds we'll consider: upper bounds, which constrain the maximum size of elements of the set, and lower bounds, which constrain the minimum size of elements of the set.

Definition 5 ***Bounded Above***
A set $A \subseteq \mathbb{R}$ is said to be bounded above if there exists a scalar $M \in \mathbb{R}$ such that for all $x \in A$, it is true that:

$$x \leq M \tag{1.154}$$

In this case, M is known as an upper bound for the set.

We may similarly define what it means for a set to be bounded below.

Definition 6 ***Bounded Below***
A set $A \subseteq \mathbb{R}$ is said to be bounded below if there exists a scalar $U \in \mathbb{R}$ such that for all $x \in A$, it is true that:

$$U \leq x \tag{1.155}$$

In this case, U is known as a lower bound for the set.

Note that it's entirely possible for both the upper and lower bounds of a set to be negative numbers! If a set is both bounded above and bounded below, it is said to be **bounded**.
Although these definitions are useful, in many cases we want to choose the

tightest possible bounds for our set. Why is this? Let's consider a simple numerical example. If we have the set of numbers:

$$A = \{1, 2, 3, 4, 5\} \subseteq \mathbb{R} \tag{1.156}$$

The numbers 5 and 1000 are both equally valid upper bounds for the set! However, since 5 is much closer to the actual values of the set, and is a *tighter* bound, it is a much more informative number than 1000.

Above: -1 and -5 are both lower bounds for the set $A \subseteq \mathbb{R}$, but -1 is a much tighter bound.

As the tightest possible upper and lower bounds of a set are often the most informative, mathematicians defined two important quantities, known as the **supremum** and **infimum**.

Definition 7 ***Supremum***
If a set $A \subseteq R$ is bounded above, the supremum of A, denoted:

$$\sup A \tag{1.157}$$

Is defined to be the smallest upper bound of A. If M is any other upper bound of A, this means that:

$$\sup A \leq M \tag{1.158}$$

Since $\sup A$ is less than or equal to *any other* upper bound of the set, it must be the tightest possible upper bound on the values of the set. Note that for the supremum of a set to be well-defined, the set must be bounded above! If we can't find an upper bound for the set, we won't be able to define its supremum. Just like we defined the tightest possible upper bound, we may define the tightest possible lower bound of a set.

Definition 8 ***Infimum***
If a set $A \subseteq R$ is bounded below, the infimum of A, denoted:

$$\inf A \tag{1.159}$$

Is defined to be the greatest lower bound of A. If U is any other lower bound of A, this means that:

$$U \leq \inf A \tag{1.160}$$

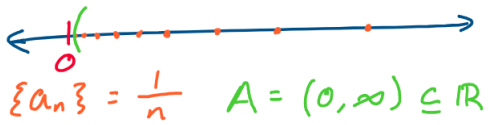
Just like we required that a set be bounded above for a supremum to exist, we require that a set be bounded below for an infimum to exist! If we can't find a lower bound for the set, there's no way we can find the greatest possible lower bound.
It's important to note that the suprema and infima[1] of sets *might not* actually be elements of the set! For example, consider the set:

$$A = (-1, 1) \subseteq \mathbb{R} \tag{1.161}$$

This is the interval from -1 to 1, not including the endpoints, on the real number line. We can see that -1 is the infimum of A and 1 is the supremum of A, yet neither are actually elements of the set! This is an important fact to keep in mind as we proceed.

1.2.4 Closed & Compact Sets

Now that we've discussed some important results regarding sets and their bounds, we can talk about two important classes of sets known as **closed sets** and **compact sets**. Here, we'll keep our discussion on the surface level and avoid formally defining a compact set in the most general sense. Interested readers are encouraged to explore their full properties in the aforementioned texts on real analysis!
The definitions of these two sets are closely tied to something called a **limit point**. Let's start our discussion there! In $\mathbb{R}^n$, a limit point of a set $A \subseteq \mathbb{R}^n$ is a point that is the limit of some non-constant sequence that's entirely contained within the set. For example:



Above: 0 *is a limit point of the set* $(0, \infty)$

In the set $A = (0, \infty) \subseteq \mathbb{R}$, 0 is a limit point of the set! This is because the sequence $\{a_n\} = \frac{1}{n}$, which is entirely contained in A, converges to 0 in the limit as $n \to \infty$!
This example brings up an interesting idea regarding limit points. Even though 0 is a limit point of the set $A = (0, \infty)$, by definition 0 is *not* included in the set! This means that sets do not necessarily contain all of their limit points.
What types of sets *do* contain all of their limit points?

Definition 9 ***Closed Set***
A set $A \subseteq \mathbb{R}^n$ *is closed if it contains all of its limit points.*

[1] The plural of supremum is *suprema* and the plural of infimum is *infima*.

If the limits of all possible sequences contained within the set are elements of the set, the set is said to be closed. What does this definition look like in practice? One consequence of this definition is that, if bounded, a closed set will include all of its endpoints. This means that sets such as:

$$A = [-1, 5] \tag{1.162}$$
$$B = [-15, -10] \cup [15, 20] \tag{1.163}$$

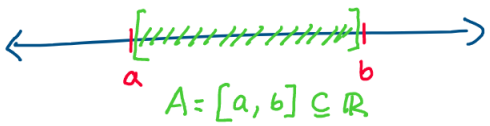
Will be closed sets, but other sets, such as:

$$C = (0, \infty) \tag{1.164}$$
$$D = [-4, 4) \tag{1.165}$$

Are *not* closed sets![2] Notice how in the second example, even though D includes one of its endpoints, -4, since it doesn't include its second endpoint, 4, it is not considered a closed set.
Are there any stronger conditions we can impose on sets other than closedness? If we look closer at the definition of a closed set, we notice that closed sets are *not* necessarily bounded! This leads us to the definition of a compact set in $\mathbb{R}^n$.

Definition 10 ***Compact Set***
A set $A \subseteq \mathbb{R}^n$ is compact if and only if it is closed and bounded.

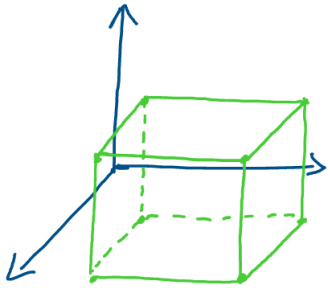
Compact sets may be thought of as a stronger version of closed sets, and enable many powerful results in analysis.
By this definition, as long as a set in $\mathbb{R}^n$ is closed, and has an upper and lower bound, it is considered compact! It's important to note that although this definition of compactness is valid in $\mathbb{R}^n$, it is not the general definition that's used in more arbitrary spaces.[3] For our purposes, however, where we largely contain our results to $\mathbb{R}^n$, it will be sufficient.
As this definition is somewhat abstract, let's think about a few examples of compact sets to gain a little bit of intuition for their form. On the real line, a classic example of a compact set is a closed interval, which has the form $[a, b]$, where $a \leq b$ and a, b are finite. This set is closed (it includes all of its limit points including its endpoints) and is bounded, since a and b are both finite!



Above: A closed interval $A = [a, b] \subseteq \mathbb{R}$.

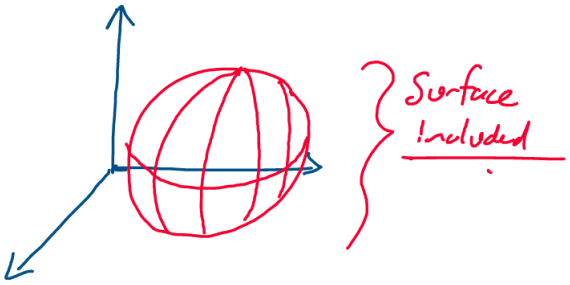
[2] $(0, \infty)$ is referred to as an *open set*! Open and closed sets are not actually opposites, despite what the naming convention would suggest! A set can be neither open nor closed.
[3] Notably, this definition of compactness is invalid on the space of L^2 functions. More generally, a set is compact if every open covering of the set has a finite subcover.

What about compact sets in $\mathbb{R}^n$? If a segment of the real line is a compact set in $\mathbb{R}$, it might be reasonable to expect that a *box* which includes all of the points on its faces might be compact in $\mathbb{R}^n$!



Above: A box with finite dimensions that includes its sides is compact in $\mathbb{R}^3$.

As we can see, this set is bounded, since it has sides of finite length. Additionally, since it contains all of the points on its faces, we can prove that this set is closed.[4] Another typical example of a compact set in n dimensions is a closed ball. This set is a ball in n dimensions that *includes* its surface, as opposed to the open balls we discussed earlier.



Above: A ball of finite radius whose surface is included is a compact set.

These are just a few examples of compact sets. Note that this list is by no means exhaustive, and the total number of compact sets is infinite!

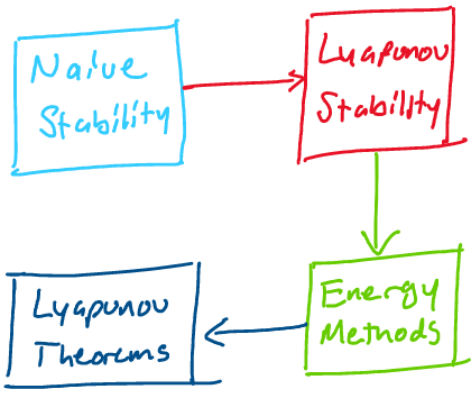
[4] This type of set is known as a k-cell mathematical literature.

1.3 Stability

Let's take a moment to discuss our progress thus far in the study of dynamical systems, and think about what areas we wish to explore next. So far, we've talked about various representations of continuous and discrete time dynamical systems, and have performed a brief analysis of properties such as equilibria and existence and uniqueness of solutions.

Now that we've established conventions for the description of these systems, we can begin to explore the theory associate with them at a deeper level, and ask new questions regarding the evolution of these systems with the passage of time. A question fundamental to the study of dynamical systems that has far reaching consequences in control theory is that of *stability*. How can we tell if a particular state in a system is stable? How can we formally define this stability? Are there any analytical tools we can use to determine stability?

In this section, we'll answer these questions through an exploration of fundamental topics in both linear and nonlinear stability. Along the way, we'll build important foundations that we'll continue to use in our study of control theory.



Above: The path we'll take in this section.

Stability is a challenging topic, so it'll prove helpful to get a sense of where we are in the development process as we move along. Let's take a moment to discuss our roadmap for this section. We'll begin by developing an intuition for what it means for something to be stable, in our discussion of naive stability. Following this, we'll make these intuitions mathematically sound with the definition of Lyapunov stability. Then, we'll learn how to apply these definitions with energy-based methods, and finish up by enjoying the power of the Lyapunov stability theorems.

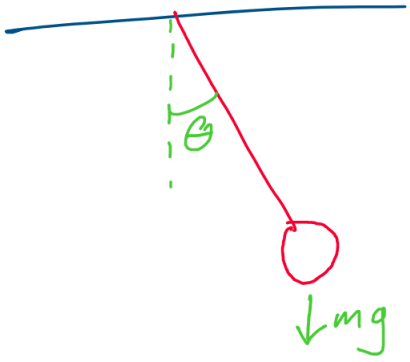
Proofs in this section

Note that this section has a more mathematical character than others we'll cover in this course. In some of the results we'll prove in this section, we'll use some more advanced techniques from real analysis. As such, the proofs in this section

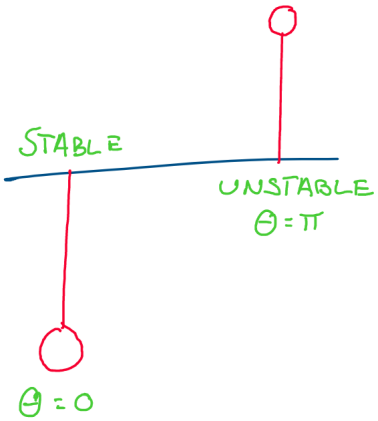
are optional, but you're highly encouraged to try them to gain the best possible appreciation for the material!

1.3.1 Naive Stability

Before we begin a formal mathematical treatment of stability, it's important that we have an intuitive idea for what it means for a point in a system to be stable. As such, before introducing any formal mathematics, we'll think about a physical example. Consider the simple pendulum below, which swings with some friction under the force of gravity.



From our knowledge of equilibrium points, we know that a pendulum has two equilibrium points, where it's positioned vertically up and vertically down with zero angular velocity.



Above: The two equilibrium points of a pendulum.

Let's think about the differences in the behavior of the pendulum at the two equilibrium points.
Physically, we know that if we pick up the pendulum, move it slightly away from its lower equilibrium point, and let go, the pendulum will remain close

to the lower equilibrium point! It will swing in some region around the lower equilibrium point and eventually settle back down to the point due to friction.
What about the upper equilibrium point? If we pick up the pendulum, move it close to its upper equilibrium point, and let go, the pendulum will immediately swing down and move far away from the upper equilibrium point!
This concept of remaining close to and straying far from equilibrium points is central to the definition of stability. If we start near an equilibrium point and stay near it afterwards for all time, that equilibrium point is said to be **stable**. If we start near an equilibrium point and continually move away from it as time goes on, that equilibrium point is said to be **unstable**.
Using the pendulum example, we can define the lower equilibrium point to be stable, as starting near to the lower equilibrium point meant that we stayed near for all time. On the other hand, the upper equilibrium point would be classified as unstable, as starting near to the upper equilibrium point did *not* mean that we would stay near for all time. Rather, the pendulum immediately swings far away from the upper equilibrium point.
Let's introduce a little bit of mathematical terminology to firm up this concept. Note that later in this section, we'll return to the definition of stability and treat it with full mathematical formality.
For the purposes of our introductory analysis, let's consider a general nonlinear dynamical system of the form:

$$\dot{x} = f(x,t),\ x \in \mathbb{R}^n,\ x(t_0) = x_0 \tag{1.166}$$

Where $x \in \mathbb{R}^n$ is the state vector of the system, $t \in \mathbb{R}$ is time, and x_0 is the initial condition. Note that in this case, the system dynamics, $f(x,t)$, have the potential to be *directly* dependent on time, t.
Suppose that x_e is an equilibrium point of this system. Recall that for x_e to be an equilibrium point of this nonlinear system, we require that:

$$0 = f(x_e,t) \text{ For all time } t \geq t_0 \tag{1.167}$$

If x_e is stable equilibrium point, based on our conceptual definition above, if we start with an initial condition x_0 that is *close* to x_e, we will stay close to x_e for all time after t_0. If x_e is an unstable equilibrium point, if we start with an initial condition x_0 close to x_e, we will move away from x_e as time goes on.

1.3.2 Lyapunov Stability

With the suitable mathematical and conceptual ideas now at our disposal, we're ready to tackle the problem of describing stability! Let's see if we can formalize the idea of "start close, stay close" stability using more precise mathematical language.
As in the previous section, we'll consider the stability of a general time variant nonlinear system of the form:

$$\dot{x} = f(x,t),\ x \in \mathbb{R}^n,\ t \in \mathbb{R}^+ \tag{1.168}$$

To keep our definitions concise, we will assume the system has an equilibrium point $x_e = 0$, such that $f(x_e, t) = 0$ for all time t. As it happens, this is a reasonable assumption to make.
If we wish to study the stability of a *nonzero* equilibrium point, x_e, we can transform that equilibrium point to be at zero by performing a simple change of coordinates in our system! If x represents our original coordinates for the system, we can define a new set of coordinates, x', as follows:

$$x' = x - x_e \tag{1.169}$$

Using this definition, we've transformed our equilibrium point x_e from a nonzero vector in x coordinates to the zero vector in x' coordinates. Since x_e is a constant equilibrium point, this change of coordinates has a minimal effect on the dynamics of the system! Since $\dot{x}_e = 0$:

$$\dot{x}' = \dot{x} - \dot{x}_e = \dot{x} = f(x, t) \tag{1.170}$$

Thus, the dynamics of our system are entirely preserved in x' coordinates.
In our analysis of stability in this section, where we only consider systems of the form $\dot{x} = f(x, t)$, we will *not* consider systems with an arbitrary input, u. This assumption turns out to be much less restrictive than one might expect! In most robotic systems, the input to the system is not arbitrary, but is rather determined by a something called a feedback controller, which computes inputs as a function of the state vector, x! Thus, it's a reasonable assumption that we can describe the input dynamics of the system just using the state vector, x.
If, however, you're interested in studying the stability of a system for an arbitrary input vector u, you're encouraged to read about a form of stability known as bounded input bounded output (BIBO) stability in a text such as *Feedback Systems* by Murray and Astrom.
Now that we've established our system and equilibrium point of interest, let's see how we can formally define stability.

Definition 11 ***Stability in the Sense of Lyapunov (SISL)***
The equilibrium point $x_e = 0$ of the nonlinear system:

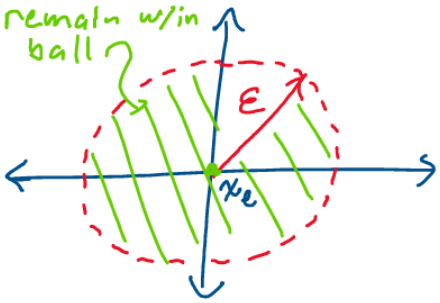
$$\dot{x} = f(x, t),\ x \in \mathbb{R}^n,\ t \in \mathbb{R}^+ \tag{1.171}$$

Is said to be stable in the sense of Lyapunov (SISL) at $t = t_0$ if for all $\varepsilon > 0$, there exists a $\delta(t_0, \varepsilon) > 0$ such that $||x(t_0)|| < \delta$ implies:

$$||x(t)|| < \varepsilon\ \forall t \geq t_0 \tag{1.172}$$

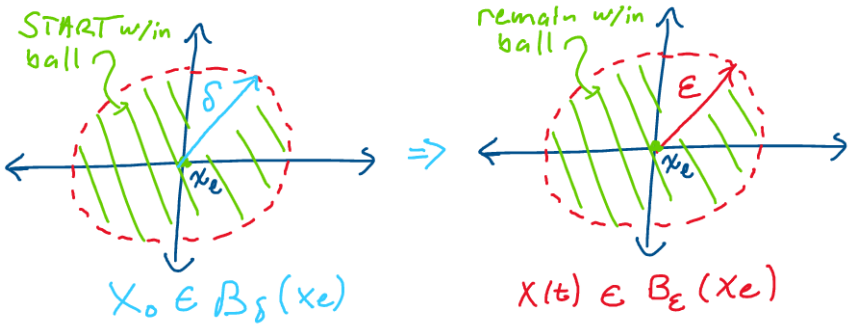
Note that the term "stable in the sense of Lyapunov" is used interchangeably with the term "Lyapunov stable."

Let's try to understand what this definition is saying, and see how it relates to our original idea of "start close, stay close" stability!
Let's imagine that we want our trajectory to stay within a distance ε of the equilibrium point x_e at all times. In other words, the trajectory must be within a *ball* of radius ε around the equilibrium point for all t.
In the two dimensional case, we can visualize this condition as:



Above: We'd like the trajectory to remain within a ball of radius ε about x_e.

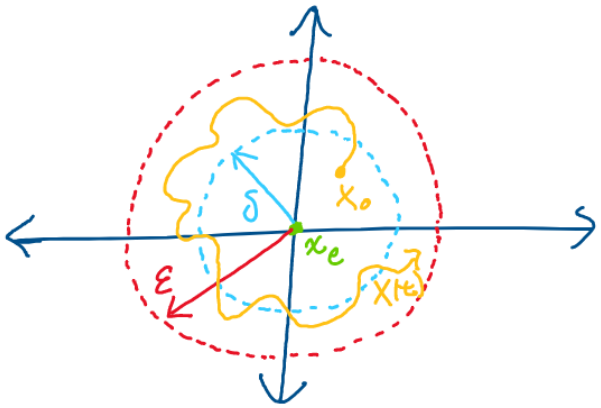
If for *any* value of ε, we can find *some* $\delta(t_0, \varepsilon)$ such that if $x(t)$ starts somewhere in a ball of radius δ around x_e, it will *stay* in a ball of radius ε for all $t \geq t_0$, then the equilibrium point is Lyapunov stable at $t = t_0$.



Above: If x_e is SISL, we can find a δ for every ε such that if we start within $B_\delta(x_e)$, we'll stay within $B_\varepsilon(x_e)$.

Why is $\delta(t_0, \varepsilon)$ a function of t_0 and ε? Since we're considering the stability of a time variant system, $\dot{x} = f(x, t)$, the conditions for stability may change as time passes, since the governing equations of the system depend on time. This is why δ is a function of t_0, and is also the reason we only conclude stability at time $t = t_0$.
What about ε? In general, if we *shrink* the radius ε that we want our trajectory to remain within, we'll also have to start within a smaller radius δ from the equilibrium point! In other words, if we require that our trajectory stays within a smaller region for all time, we'll likely need to start our system *closer* to the equilibrium point. This gives the dependence of δ on ε.

Let's reason about this with a numerical example. Suppose that to remain within a distance of $\varepsilon = 1000\ m$ from our equilibrium point, we have to start within a distance of $\delta = 100\ m$ from the equilibrium point. However, if we shrink ε to $100\ m$, we might need to start within a smaller radius, for example $10\ m$, to remain within a radius ε of our equilibrium point for all time.
We can reason about the relationship between ε and δ through the lens of a *challenge* and *response*. ε sets a challenge - a radius that our trajectory must remain within, while δ provides a response - an range of initial conditions that allow our trajectory to remain within the challenge radius ε.
Let's think more carefully about the conditions the definition of Lyapunov stability imposes on a system, and turn to the two-dimensional case to gain some visual perspective.



Above: If we start within a radius δ, we'll remain within a radius ε for all time.

We know from the definition of Lyapunov stability that if x_e is stable in the sense of Lyapunov that there exists some $\delta(t_0, \varepsilon)$ such that picking an initial condition x_0 within a ball of radius δ around the equilibrium point guarantees the trajectory of the system, $x(t)$, will remain within the larger ball of radius ε. Does this condition, however, actually guarantee that the trajectories $x(t)$ that start in this region will *converge* to the equilibrium point?
As we can see in the image above, even though it satisfies the conditions for Lyapunov stability, the trajectory $x(t)$ never actually converges to the equilibrium point, x_e - it simply swims around in the ball of radius ε.
Thus, although the basic definition of Lyapunov stability tells us if a trajectory will *stay near* to an equilibrium point, it tells nothing about whether a trajectory will actually *converge* to an equilibrium point.
For a guarantee of stability *and* convergence, we turn to the definition of asymptotic stability.

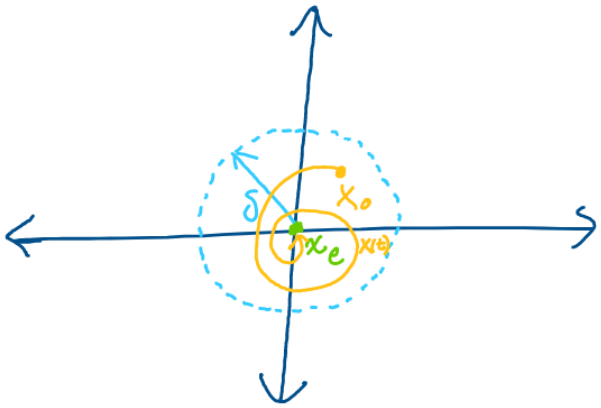
Definition 12 ***Asymptotic Stability***
The equilibrium point $x_e = 0$ of the system $\dot{x} = f(x,t), x \in \mathbb{R}^n$ is said to be

asymptotically stable at $t = t_0$ if the following two conditions are satisfied:

1. *x_e is stable in the sense of Lyapunov*
2. *There exists a scalar $\delta(t_0) > 0$ such that if $||x(t_0)|| < \delta$:*

$$\lim_{t \to \infty} x(t) = x_e = 0 \tag{1.173}$$

Thus, the only extra condition that we require for asymptotic stability is that for *some* region around the equilibrium point, all of the trajectories starting within the region will *converge* to the equilibrium point.



Above: There exists a region around the equilibrium point such that trajectories converge to the equilibrium point.

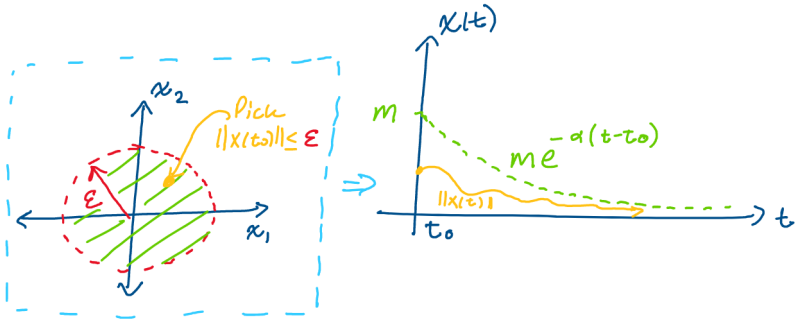
Note that although asymptotic stability ensures convergence, it doesn't impose any requirements on the *rate* of convergence! As long as $\lim_{t \to \infty} x(t) = x_e$, the convergence condition for asymptotic stability is satisfied.
Are there any forms of stability which *do* specify the rate of convergence? If we want to ask the question of whether a trajectory will converge *quickly* to an equilibrium point, we may use the definition of exponential stability.

Definition 13 ***Exponential Stability***
The equilibrium point $x_e = 0$ of the system $\dot{x} = f(x, u), x \in \mathbb{R}^n$ is said to be exponentially stable if there exist constants $m, \alpha > 0, \varepsilon > 0$ such that for all $x(t_0) : ||x(t_0)|| \leq \varepsilon$, is is true that:

$$||x(t)|| \leq m e^{-\alpha(t - t_0)} ||x(t_0)||, \ \forall t \geq t_0 \tag{1.174}$$

The largest value of α for which this is true is called the rate of convergence.

What does an exponentially stable equilibrium point look like? Once again, let's turn to the two-dimensional case, where $x(t) \in \mathbb{R}^2$, to visualize this definition. Consider the following visualization:



Above: x_e is exponentially stable if there is some region of initial conditions for which we can bound the trajectory by an exponentially stable function.

As we can see in the image above, if x_e is exponentially stable, we can find some $\varepsilon > 0$ such that if our initial condition is within a ball of radius ε around the equilibrium point, its distance from the equilibrium point will be bounded above by some exponential function for all time!
Note that *not all* asymptotically stable systems are exponentially stable! However, we can prove without too much trouble that all exponentially stable systems are asymptotically stable. Let's write out this proof to get a sense of what it means to use these definitions in practice.

Proposition 1 ***Exponential Stability Implies Asymptotic Stability***
If $x_e = 0$ is an exponentially stable equilibrium point of the system $\dot{x} = f(x,t)$ within a region $\{x \in \mathbb{R}^n : ||x|| \leq \varepsilon\}$, it is also an asymptotically stable equilibrium point of the system within that region.

Proof: Where can we begin our proof of this proposition? Let's restate the definition of exponential stability, so we know what we're working with. Recall that if $x_e = 0$ is an exponentially stable equilibrium point, there exist constants $m, \alpha > 0, \varepsilon > 0$ such that for all $||x(t_0)|| \leq \varepsilon$, we have that:

$$||x(t)|| \leq me^{-\alpha(t-t_0)}||x(t_0)||, \ \forall t \geq t_0 \tag{1.175}$$

To show that $x_e = 0$ is an asymptotically stable equilibrium point, we must prove that it is stable in the sense of Lyapunov *and* that $\lim_{t\to\infty} x(t) = 0$.
Let's start by tackling the first part of the definition, that our exponentially stable equilibrium point is stable in the sense of Lyapunov. If a point is stable in the sense of Lyapunov, for all $\varepsilon' > 0$, there exists a $\delta(t_0, \varepsilon') > 0$ such that:

$$||x(t_0)|| < \delta \implies ||x(t)|| < \varepsilon' \ \forall t \geq t_0 \tag{1.176}$$

Let's try a constructive existence proof for δ, and show such a δ exists by finding an explicit formula. Let's see if we can find some sort of relationship between δ and ε, and see where that takes us!
Let's only consider δ to be less than ε, the radius of the region where the system is exponentially stable. If $||x(t_0)|| < \delta$ and x_e is exponentially stable, what bounds on the value of $||x(t)||$ can we make? We can form the following set of inequalities:

$$||x(t)|| \leq me^{-\alpha(t-t_0)}||x(t_0)||, \ \forall t \geq t_0 \tag{1.177}$$
$$< me^{-\alpha(t-t_0)}\delta, \ \forall t \geq t_0 \tag{1.178}$$

Now, for all $t \geq t_0$, we know that the exponential term, $\exp(-\alpha(t-t_0)) \leq 1$ by properties of the exponential function! Thus, we may form another inequality:

$$||x(t)|| < me^{-\alpha(t-t_0)}\delta, \ \forall t \geq t_0 \tag{1.179}$$
$$\leq m\delta, \ \forall t \geq t_0 \tag{1.180}$$

Now, we remember that our original goal was to find a $\delta(t_0, \varepsilon')$ such that $||x(t)||$ can be bounded above by ε' for all $t \geq t_0$. Looking at the inequality above, let's define:

$$\delta = \min\left\{\frac{\varepsilon'}{m}, \frac{\varepsilon}{m}\right\} \tag{1.181}$$

This choice will ensure that at our initial condition, the exponential bound will always hold. Then, using the inequalities we formed above, for all $||x(t_0)|| < \delta$, we have for all values of ε':

$$||x(t_0)|| < \delta \implies ||x(t)|| < m\delta \leq m\frac{\varepsilon'}{m} = \varepsilon', \ \forall t \geq t_0 \tag{1.182}$$

Thus, we have shown that *every* exponentially stable equilibrium point is stable in the sense of Lyapunov within the region where exponential stability holds!
To complete our proof that exponential stability implies asymptotic stability, we must now show that our exponentially stable equilibrium point satisfies the second condition of asymptotic stability, that for some set of initial conditions, $\lim_{t\to\infty} x(t) = 0$.
We know that within radius ε of the equilibrium point:

$$0 \leq ||x(t)|| \leq me^{-\alpha(t-t_0)}||x(t_0)||, \ \forall t \geq t_0 \tag{1.183}$$

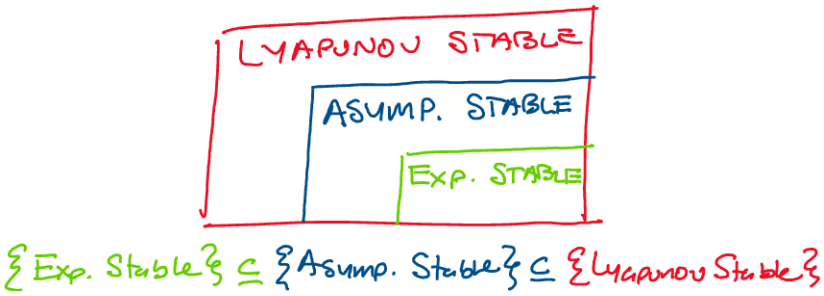
Taking the limits as $t \to \infty$ of each piece of the inequality:[5]

$$\lim_{t\to\infty} 0 \leq \lim_{t\to\infty} x(t) \leq \lim_{t\to\infty} me^{-\alpha(t-t_0)}||x(t_0)|| \tag{1.184}$$
$$0 \leq \lim_{t\to\infty} x(t) \leq 0 \tag{1.185}$$

[5] More formally, this is an application of the squeeze theorem from calculus.

Thus, we conclude that $\lim_{t\to\infty} x(t) = 0$ as long as $||x(t)|| \leq \varepsilon$. Therefore, our exponentially stable equilibrium point satisfies the second requirement for asymptotic stability. This completes the proof! □

Let's take a moment to review how these three types of stability relate to each other. We now know that exponential stability is the strictest form, followed by asymptotic, and then stable in the sense of Lyapunov.



Above: The hierarchy of three different types of stability.

Global Stability

How else can we strengthen these conditions of stability? If we look closely at our definitions of Lyapunov, asymptotic, and exponential stability, we find that all three definitions are only *local* results - that is, they only hold for certain regions around the origin!

In the definition of asymptotic stability, for example, we only require that the $\lim_{t\to\infty} ||x(t)||$ is x_e within a certain radius δ of the equilibrium point. Outside of this region, we have no guarantees on the limit of the trajectory! This same remark can be made for exponential stability, where we only require $||x(t)||$ be bounded above by an exponential function in some region of radius ε around the origin.

What consequences do these statements have? The fact that we only required our definitions above to hold within some radius δ or ε of the equilibrium point means that our definitions of stability thus far only describe the stability of the system close to the equilibrium point!

An equilibrium point is said to be **globally stable** if it is stable for *all* initial conditions of the system! Using more mathematical notation, if x_e is globally stable, it is stable for *all* initial conditions $x_0 \in \mathbb{R}^n$. Since global stability guarantees stability for all initial conditions, it is a much more powerful condition than local stability.

Uniform Stability

Let's think about another way we can make our definitions of stability stronger! When we look at the basic definition of Lyapunov stability, we notice that the range of initial conditions, $\delta(t_0, \varepsilon)$ is *dependent* on the starting time of the

system! This is because the system we wish to analyze:

$$\dot{x} = f(x, t) \tag{1.186}$$

Is *explicitly* dependent on time! If the dynamics of the system change as time passes, there is the potential that an equilibrium point stable at the time t_0 might *not* be stable at another starting time t_0'!
With this in mind, let's develop a more restrictive definition of stability that guarantees stability of an equilibrium point for all time.

Definition 14 ***Uniform Stability***
The equilibrium point x_e of the system $\dot{x} = f(x, t)$ is said to be uniformly stable if for all $\varepsilon > 0$, there exists a $\delta(\varepsilon)$, not dependent on time such that for all initial conditions and all starting times t_0, if $||x(t_0)|| < \delta$, it is true that:

$$||x(t)|| < \varepsilon \ \forall t \geq t_0 \tag{1.187}$$

Uniform stability only makes a small modification of our original definition of stability in the sense of Lyapunov. Here, we insist that δ is *only* a function of ε, and not a function of time. This means that the passing of time does not impact our ability to find a δ for the system, and therefore does not impact the stability of the equilibrium point.
Similarly, we can define a uniform version of asymptotic stability.

Definition 15 ***Uniform Asymptotic Stability***
The equilibrium point x_e of the system $\dot{x} = f(x, t)$ is said to be uniformly asymptotically stable if the following two conditions are satisfied:

1. *The equilibrium point x_e is uniformly stable.*
2. *There exists a $\delta > 0$, not dependent on t_0, such that for all initial conditions $x(t_0)$ satisfying $||x(t_0)|| < \delta$, it is true that:*

$$\lim_{t \to \infty} x(t) = x_e = 0 \tag{1.188}$$

Note that we also require the convergence of limit in the second condition to be "uniform," a special type of convergence that we won't discuss here!

Let's summarize how all of these definitions of stability relate to one another. We made an initial set of definitions for the *local* stability of systems, with certain definitions requiring convergence to the equilibrium point. We then found we

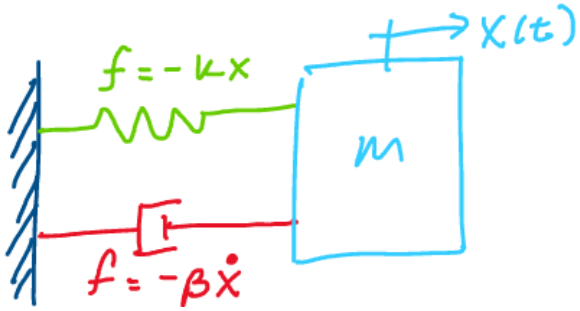
could make these results stronger by requiring them to hold *globally*. Finally, we defined a set of stability conditions that ensure the stability of a point doesn't change with time.

1.3.3 An Energetic Approach

Let's think about some challenges that might come up when working with these definitions! Suppose we want to use the formal definition of Lyapunov stability, which requires finding a $\delta(t_0, \varepsilon)$ that restricts our initial conditions and allows $x(t)$ to remain sufficiently close to the equilibrium point $x_e = 0$.
What knowledge about the system does it seem like we'll need to apply this definition? To use this definition of stability, it seems like we'll need an explicit solution to the differential equation:

$$\dot{x} = f(x, t), \; x(t_0) = x_0 \tag{1.189}$$

Only then will we be able to tell if $x(t)$ will remain close to our equilibrium point of interest for all time! But, general nonlinear differential equations are *incredibly* hard to solve, and often have no closed form solution!
To solve this problem, we want to find some way of analyzing stability *without* having to explicitly solve for $x(t)$. Let's reason about a method of doing this with a physical system. Consider the simple mass-spring-damper system:



Above: A mass oscillates on a spring with a damping force.

Note that the damper, sketched above in red, applies a force to the mass opposite and proportional to its velocity, $\dot{x}$. You can think of the damper as something that resists quick motion in the system. Assuming that $x = 0$ when the spring is unstretched, this system is described by the differential equation:

$$m\ddot{x} = -kx - \beta\dot{x} \tag{1.190}$$

Where m, k, β are positive scalar constants. We can quickly check that the state $[x, \dot{x}] = [0, 0]$ is the only equilibrium point of the system.
Without explicitly solving for the equations of motion, can we come to any conclusion about the stability of this equilibrium point? One potential lead on the answer to this question lies in the total energy of the system.
We know that for the mass-spring-damper system, energy is always greater than

zero. Thus, if we could show that the energy of the system is always decreasing, we could conclude that the energy approaches zero. When the energy of the system is zero, the mass can't be moving, and the system must be frozen! This means that when the system has zero energy, it *must* be at an equilibrium point. Let's apply this thought process to the mass-spring-damper system, and see how energy changes over time. We can express the total energy of the system as:

$$E = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}kx^2 \tag{1.191}$$

Let's take the first time derivative of energy to determine the rate of change of energy in the system:

$$\dot{E} = m\dot{x}\ddot{x} + kx\dot{x} \tag{1.192}$$

$$= (m\ddot{x} + kx)\dot{x} \tag{1.193}$$

Substituting in the dynamics of the system for $m\ddot{x}$, we get:

$$\dot{E} = (-kx - \beta\dot{x} + kx)\dot{x} \tag{1.194}$$

$$\dot{E} = -\beta\dot{x}^2 \tag{1.195}$$

Since $\dot{x}^2 \geq 0$, we conclude that the rate of change of energy, $\dot{E}$, is less than or equal to zero, and is only zero when velocity is zero! Thus, the energy of the system *decreases* as time passes and will approach zero as $t \to \infty$, making the system settle at its equilibrium point.

The idea of using the rate of change of energy to analyze the stability of a system is a very powerful one! Without solving the equations of motion for $x(t)$, we were able to make a conclusion about stability just using the rate of change of energy in the system. Can we extend the idea of using energy to study the stability to *general* nonlinear systems $\dot{x} = f(x, t)$?

In the next section, we'll introduce a general class of functions that act *just like* energy functions, but for arbitrary nonlinear systems! We'll then apply these special functions to describe the stability of general nonlinear systems.

1.3.4 Energy-Like Functions

In this section, we'll build up a class of functions that act just like energy functions for arbitrary systems! We'll think about the core properties required of these functions in a mathematically precise manner. Note that in this section, we'll use some terminology we previously developed in the section on mathematical preliminaries.

Let's begin by discussing an important mathematical operator known as the **Cartesian product**, which we'll make frequent use of in our definitions in this section!

Definition 16 ***Cartesian Product*** *Let $A = \{a_1, a_2, ...,\}, B = \{b_1, b_2, ...\}$ be two sets. The Cartesian product of A and B, denoted $A \times B$, is the set of all combinations of elements of A and B.*

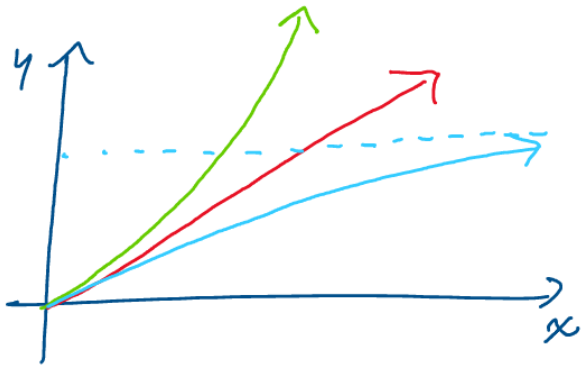
$$A \times B = \{(a_1, b_1), (a_1, b_2), (a_1, b_3), ...\} \tag{1.196}$$

Note that although the Cartesian product shares the symbol $\times$ with the vector cross product, it refers to a different fundamental concept. Whenever $\times$ is applied to two (or more) sets, think of the Cartesian product, and when $\times$ is applied to two vectors, think of the vector cross product. Keep an eye out for the Cartesian product in describing the domains of different functions as we proceed!
Let's now turn out attention to describing different classes of functions, and begin by introducing a simple yet fundamental class.

Definition 17 ***Strictly Increasing Function***
A function $f(x) : \mathbb{R} \to \mathbb{R}$ is said to be strictly increasing if for all $x, y \in \mathbb{R}$, if $x < y$, it is true that:

$$f(x) < f(y) \tag{1.197}$$

If a function is strictly increasing, the further we get away from the origin, the higher the value of the function will be! Note that despite how the definition may initially sound, it is *not* necessarily true that a strictly increasing function will grow to infinity!



Above: Three strictly increasing functions.

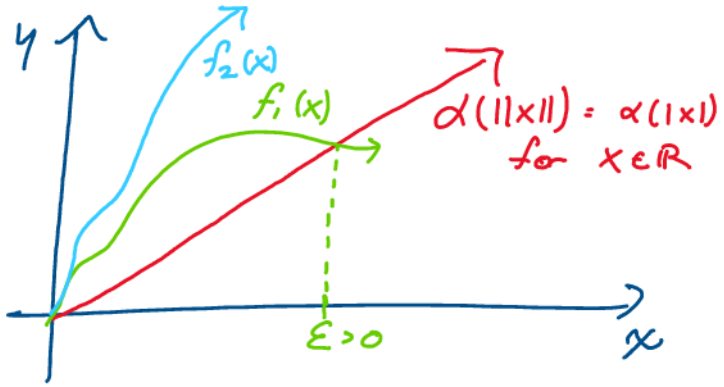
For instance, if a function is asymptotic to a certain value, such as the light blue function in the image above, it can be strictly increasing yet not tend to

infinity! We can use these strictly increasing functions to define our first type of energy-like function for an arbitrary system!

Definition 18 ***Locally Positive Definite Function (LPDF)***
A continuous function $V(x,t) : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}$ *is said to be a locally positive definite function (LPDF) if for some* $\varepsilon > 0$*, there exists a strictly increasing function* $\alpha : \mathbb{R}^+ \to \mathbb{R}$ *such that* $\alpha(0) = 0$ *and:*

1. $V(0,t) = 0$ *for all* $t \in \mathbb{R}^+$
2. $V(x,t) \geq \alpha(||x||)$ *for all* $x \in B_\varepsilon(0)$ *and all* $t \geq 0$

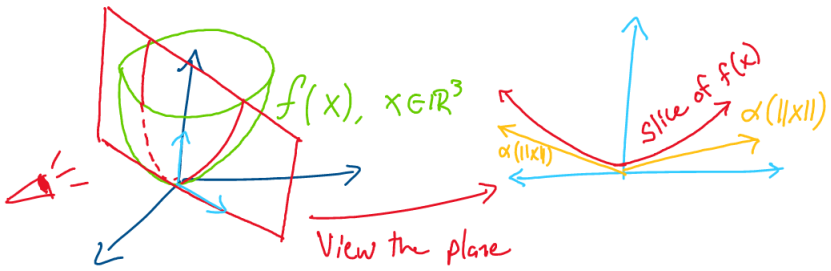
Let's think about the different parts of this definition. Firstly, we defined a function $V(x,t) : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}$ - what does this mean? This means that the function takes in a combination of a vector $x \in \mathbb{R}^n$ and a positive scalar $t \in \mathbb{R}^+$, and returns a scalar. Recall from our discussion of the Cartesian product that the notation $\mathbb{R}^n \times \mathbb{R}^+$ refers to the set of all combinations of n dimensional vectors and positive scalars.
Next, let's discuss the two conditions of the definition, which only need to be true locally. It's in these conditions that we can see the similarities between the class of locally positive definite functions and energy functions for physical systems! Firstly, $V(0,t) = 0$ for all t - no matter what the value of t is, as long as x is at the origin, the value of the function is zero. If we think back to our mass-spring-damper system, we recall that the energy of the system was zero at the origin, $[x, \dot{x}] = [0, 0]$.
The second condition is somewhat more subtle - let's see if we can visualize this constraint! Let's begin by considering the case where V *isn't* a function of t, and is simply a function of a scalar $x \in \mathbb{R}$. Sketching out a few possibilities:



Above: f_1 *and* f_2 *are locally bounded above by a strictly increasing function.*

In this simple case, as long as a function is greater than a strictly increasing function for *some* region around the origin, the second condition is satisfied!

Notice that the functions themselves *don't* need to be strictly increasing or have particularly regular behavior for this to be true - they just need to be bounded below by some strictly increasing function.
Let's try and visualize this condition in higher dimensions! As it happens, the conditions enforce more or less the same behavior as the one dimensional case. Let's consider the case where $x \in \mathbb{R}^2$ as an example.

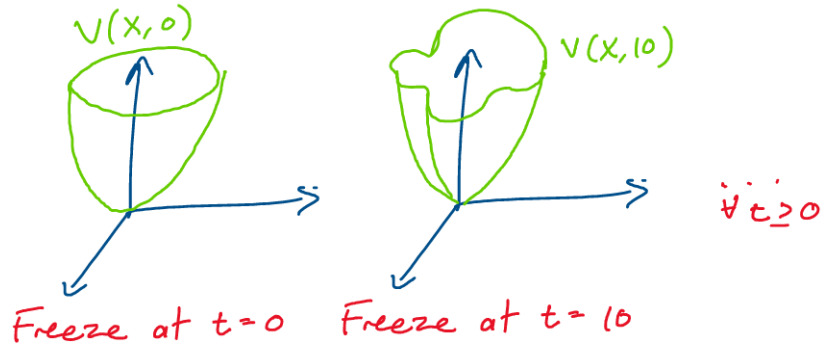


Above: We can visually interpret this condition in higher dimensions by taking slices of our function.

Visually, we can reason about the condition:

$$V(x,t) \geq \alpha(||x||) \tag{1.198}$$

In higher dimensions by slicing the surface of our higher dimensional function with a plane, as in the image above. By looking at the behavior of the function in each slice, we can reduce the higher dimensional problem back into the one dimensional problem we initially looked at!
If for every possible slice of our function, the slice appears to be bounded below by a strictly increasing function within a certain region of the origin, the condition $V(x,t) \geq \alpha(||x||)$ is satisfied.
What effect does changing t have? Visually, we can imagine something being true for all t if we freeze the value of the function $V(x,t)$, for each t, and make sure the function satisfies the required conditions.



Above: we can imagine the $\forall t$ condition as freezing the appearance of $V(x,t)$ at each t and checking that it is well-behaved.

How does the second condition, that $V(x,t) \geq \alpha(||x||)$, relate to the idea of energy? Let's think about this condition in the context of the mass-spring-damper once again.
If we move the vector $[x, \dot{x}]$ far away from the origin, we know two things will happen. Firstly, if we increase x, we'll gain a high amount of potential energy from stretching the spring away from its equilibrium point. Secondly, if we increase $\dot{x}$, we'll gain kinetic energy, from increasing the speed of the mass. Thus, as we increase $|||[x, \dot{x}]|||$, the energy of the system will be bounded below by a strictly increasing function of $|||[x, \dot{x}]|||$.
Using our visualizations as inspiration, is can we simplify the definition of a locally positive definite function? As it happens, there is an alternate characterization of these functions that will often be easier to work with.

Proposition 2 ***Alternate Characterization of LPDFs***
A continuous function $V(x,t) : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}$ is locally positive definite if:

1. *$V(0,t) = 0$ for all $t \in \mathbb{R}^+$.*
2. *There exists an $\varepsilon > 0$ such that $V(x,t) > 0$ for all nonzero $x : ||x|| \leq \varepsilon$ and all $t \in \mathbb{R}^+$.*

Let's extend the definition of a locally positive definite function to be global. Instead of just acting like an energy function locally, the following type of function will act like an energy function *everywhere* in $\mathbb{R}^n$!

Definition 19 ***Positive Definite Function (PDF)***
A function $V(x,t) : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}$ is a positive definite function if there exists a strictly increasing function $\alpha : \mathbb{R}^+ \to \mathbb{R}$ such that $\alpha(0) = 0$ and:

1. *$V(0,t) = 0$ for all $t \in \mathbb{R}^+$*
2. *$V(x,t) \geq \alpha(||x||)$ for all $x \in \mathbb{R}^n$ and all $t \geq 0$*
3. *$\lim_{p \to \infty} \alpha(p) = \infty$*

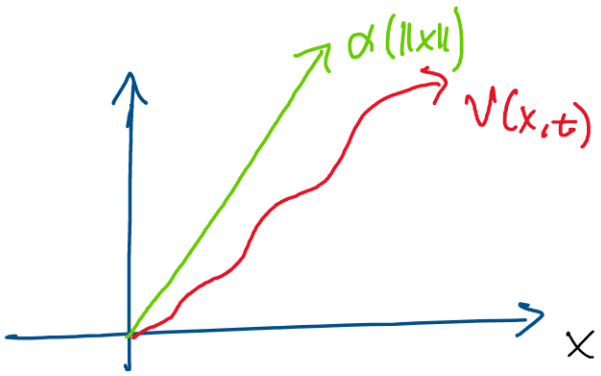
Let's review the differences between a positive definite function and a locally positive definite function. For a positive definite function, instead of the first two conditions being required to be true in *some region* around the origin, they are required to be true *globally* (for all values of x).
Secondly, we have the additional requirement that our increasing function $\alpha(p)$ *must* grow to ∞ as $p \to \infty$. This requirement ensures that our strictly increasing function will not asymptotically approach a constant value as we increase the

value of $||x||$ to infinity.
The reason we didn't include this condition in the local definition is that in a local region, we can't extend anything to infinity - this is only a concern for a global result.
Now that we've performed an extensive discussion of functions that are *bounded below* by another function, let's think about a type of function that is bounded above!

Definition 20 ***Decrescent Function***
A continuous function $V(x,t) : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}$ is said to be decrescent if for some $\varepsilon > 0$, there exists a strictly increasing function $\beta : \mathbb{R}^+ \to \mathbb{R}$ such that $\beta(0) = 0$ and:

$$V(x,t) \leq \beta(||x||) \; \forall x \in B_\varepsilon(0), \; \forall t \geq 0 \tag{1.199}$$

As we can see from the definition above, instead of being bounded below by a strictly increasing function that is zero at the origin, a decrescent function is *bounded above*. Notice that this does *not* imply a decrescent function is decreasing! For instance, consider the function:



Above: A decrescent function, in red, is bounded above by a strictly increasing function, in green.

The fact that a decrescent function is bounded above simply ensures that it will decrease *uniformly* as we decrease $||x||$ to zero. This is where the name "decrescent" comes from.
Before we apply these definitions to the study of stability, we have one more type of function to consider! You may have noticed that in many of these definitions, we relied on some sort of strictly increasing scalar function that is zero at the origin. Since this is such a common class of function in the study of stability, we make the following definition:

Definition 21 ***Class $\mathcal{K}$ Function***
A class $\mathcal{K}$ function is a continuous function $f(x) : \mathbb{R}^+ \to \mathbb{R}$ that is strictly increasing and equal to zero at the origin.

This makes the functions α and β used in the definitions above class $\mathcal{K}$ functions. In addition to being used in the study of stability, these functions also notably pop up in the field of safety critical control, which we'll explore briefly later on in the course.

1.3.5 Lyapunov Stability Theorems

After developing our definitions of Lyapunov stability and classifying various energy-like functions, we're finally ready to tackle the problem of proving stability for nonlinear systems.

The Direct Method of Lyapunov

Recall that when studying the mass-spring-damper system, we concluded that, since the time derivative of energy was less than or equal to zero for all states, the system would eventually settle around its equilibrium point.
For a general nonlinear system, $\dot{x} = f(x, t)$, we may define a measure of system energy using a scalar, energy-like function known as a **Lyapunov function**. In general, Lyapunov functions are denoted:

$$V(x, t) : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R} \tag{1.200}$$

$V(x, t)$ takes in a combination of the current state vector of the system, $x \in \mathbb{R}^n$, and the current time in the system, $t \in \mathbb{R}^+$, and returns a single scalar value $V(x, t) \in \mathbb{R}$ that represents the energy of the system at state x and time t.
Earlier, when dealing with the energy of the mass-spring-damper system, we found it productive to take the time derivative of the energy of the system. We found that when we substituted the system dynamics into the expression for the derivative of energy, we were able to show that energy was decreasing. In other words, when we took the time derivative of energy and applied the *constraint* of the system dynamics, we found that energy was decreasing!
For a general nonlinear system, how can we take the first time derivative of a Lyapunov function $V(x, t)$ subject to the dynamics $\dot{x} = f(x, t)$?

Definition 22 ***Derivative Along a Trajectory***
The first time derivative of the function $V(x, t) : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ along the trajec-

tories of the system $\dot{x} = f(x,t)$ is defined:

$$\dot{V} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial x} f(x,t) \tag{1.201}$$

Where does this definition come from? By the chain rule, we may evaluate the time derivative of $V(x,t)$ as:

$$\frac{dV(x,t)}{dt} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial x}\frac{dx}{dt} \tag{1.202}$$

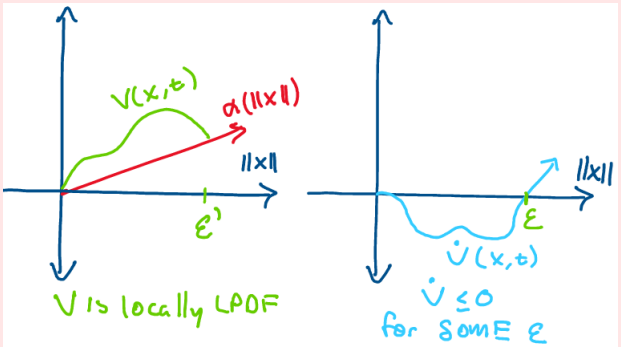
Now, if we want to add the constraint that we're *evaluating* the time derivative of V subject to the constraint of the system dynamics, $\dot{x} = f(x,t)$, all we need to do is replace the derivative $\dot{x} = \frac{dx}{dt}$ with the system dynamics! This leaves us with our definition:

$$\frac{dV}{dt} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial x} f(x,t) \tag{1.203}$$

Which must be the value of $\dot{V}$ subject to the dynamics of our system.
Now that we have a method of taking the derivative of a function subject to the constraint of the system dynamics $\dot{x} = f(x,t)$, we may state the basic theorem of Lyapunov, also known as direct method of Lyapunov.

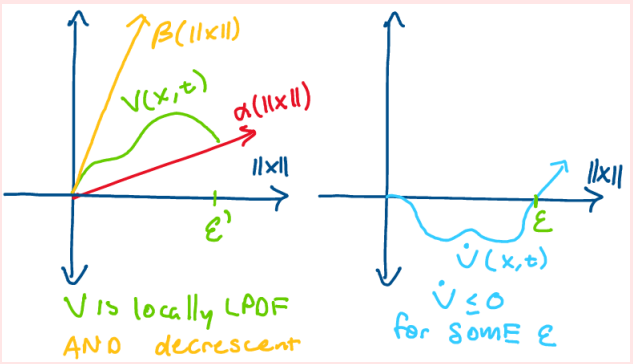
Theorem 2 ***Basic Theorem of Lyapunov (Direct Method)***
Suppose $\dot{x} = f(x,t)$ is a system with an equilibrium point $x_e = 0$. Let $V(x,t) : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}$ be a Lyapunov function, with derivative $\dot{V}$ along the trajectories of the system $\dot{x} = f(x,t)$. Using $V, \dot{V}$, we may establish the following forms of stability and their conditions:

1. ***Locally SISL**: If $V(x,t)$ is locally positive definite and there exists some $\varepsilon > 0$ such that $\dot{V}(x,t) \leq 0$ for all $x \in B_\varepsilon(0)$ and for all t, then $x_e = 0$ is a locally stable equilibrium point in the sense of Lyapunov.*



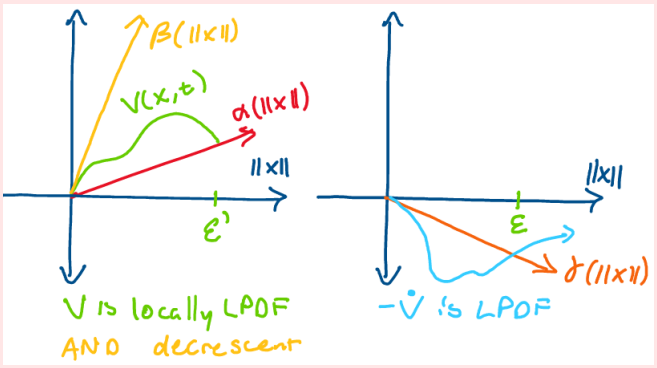
Above: V must be LPDF and $\dot{V} \leq 0$ locally.

2. ***Locally Uniformly SISL***: *If $V(x,t)$ is locally positive definite and decrescent, and there exists some $\varepsilon > 0$ such that $\dot{V}(x,t) \leq 0$ for all $x \in B_\varepsilon(0)$ and for all t, then $x_e = 0$ is a locally uniformly stable equilibrium point in the sense of Lyapunov.*



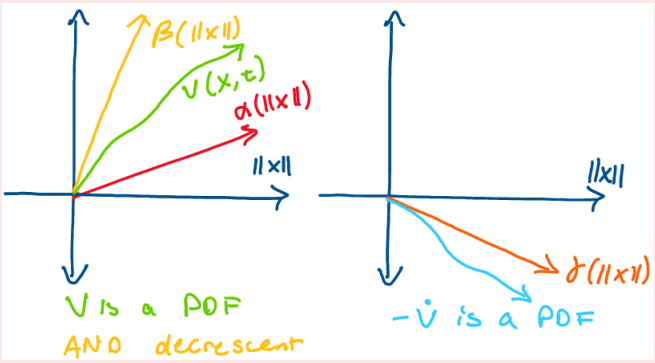
Above: V must be LPDF and decrescent and $\dot{V} \leq 0$ locally.

3. ***Locally Uniformly Asymptotically Stable***: *If $V(x,t)$ is locally positive definite and decrescent, and $-\dot{V}(x,t)$ is locally positive definite, then $x_e = 0$ is a locally uniformly asymptotically stable equilibrium point.*



Above: V must be LPDF and decrescent and $-\dot{V}$ LPDF.

4. ***Globally Uniformly Asymptotically Stable***: *If $V(x,t)$ is positive definite and decrescent, and $-\dot{V}(x,t)$ is positive definite, then $x_e = 0$ is a globally uniformly asymptotically stable equilibrium point.*



Above: V must be PDF and decrescent and $-\dot{V}$ PDF.

As a brief point of interest regarding this theorem, notice how the constraint of positive definite *and* decrescent forces a Lyapunov function to converge smoothly to zero as $||x||$ approaches zero. This is what affords us the extra condition of uniform stability.

Using this powerful theorem, we observe that by using a Lyapunov function, we're able to describe the stability of equilibrium points of arbitrary nonlinear systems *without* explicitly having to solve for the solution $x(t)$ to a differential equation! This highlights the power of using an energy-based approach to describe general nonlinear systems.

Before we practice applying this result, let's discuss one more important stability theorem. In the theorem above, although we characterized four important types of stability, we didn't discuss the conditions for *exponential stability*! Can we reason about the exponential stability of an equilibrium point using a Lyapunov function?

Theorem 3 ***Exponential Stability Theorem***
Suppose $\dot{x} = f(x,t)$ is a system with an equilibrium point $x_e = 0$. $x_e = 0$ is a locally exponentially stable equilibrium point if and only if, for some $\varepsilon > 0$, there exists a Lyapunov function $V(x,t)$ such that for all t:

$$\alpha_1||x||^2 \leq V(x,t) \leq \alpha_2||x||^2 \tag{1.204}$$

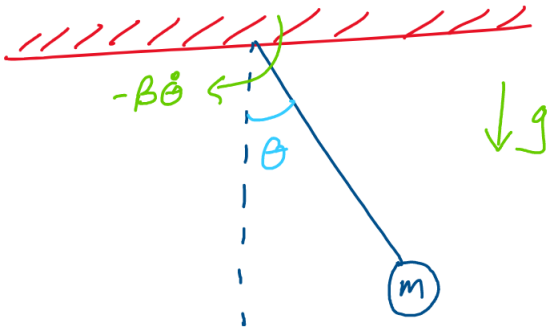
$$\dot{V} \leq -\alpha_3||x||^2 \tag{1.205}$$

$$\left|\left|\frac{\partial V}{\partial x}(x,t)\right|\right| \leq \alpha_4||x|| \tag{1.206}$$

For some positive constants α_i and for all x such that $||x|| \leq \varepsilon$. If these conditions hold for all $x \in \mathbb{R}^n$, $x_e = 0$ is a globally exponentially stable equilibrium point.

Notice how when we move from the less restrictive stability definitions to the more restrictive, the number and strictness of conditions on the Lyapunov function also increase! Recall that in general, exponential stability is the *strongest* form of stability an equilibrium point can have. Consequently, it's also the most challenging to achieve!

Let's get some practice using the direct method of Lyapunov on a simple system. Consider the pendulum of mass m and length l, whose angle to the vertical is described by θ. Let's see what we conclude about the stability of the equilibrium point $(\theta, \dot{\theta}) = (0, 0)$.



Above: A pendulum swings with a frictional torque under the force of gravity.

Suppose the pendulum swings under the force of gravity, and is affected by a frictional torque proportional to its velocity, $-\beta\dot{\theta}$ ($\beta > 0$), which is applied at its point of rotation.

Using the methods of Newtonian or Lagrangian dynamics, we may show that the motion of the pendulum is governed by the differential equation:

$$ml^2\ddot{\theta} = -mgl\sin\theta - \beta\dot{\theta} \tag{1.207}$$

To analyze the stability of this system, we first need to put it in standard state space form. We define a state vector x using the phase variable convention:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \tag{1.208}$$

Using x, we rewrite the system dynamics in state space form as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{g}{l}\sin x_1 - \beta x_2 \end{bmatrix} \tag{1.209}$$

$$\tag{1.210}$$

Note that in this case, the dynamics of the pendulum do not change with time, so our system is simply $\dot{x} = f(x)$ instead of $\dot{x} = f(x, t)$. Now, let's try to find some locally positive definite Lyapunov function $V(x)$ that is a function of our state vector!

Let's try using the energy of the system! We may calculate the total energy of the pendulum by adding the kinetic and potential energies:

$$E = \frac{1}{2}ml^2\dot{\theta}^2 + mgl(1 - \cos\theta) \tag{1.211}$$

$$= \frac{1}{2}ml^2x_2^2 + mgl(1 - \cos x_1) \tag{1.212}$$

Let's make a few simplifications to this function to make our lives a little easier. We know that if we scale any locally positive definite function by a positive scalar, it should still be locally positive definite!
Currently, there are a lot of constants floating around in the expression for energy - to simplify our analysis, we'll divide our expression for energy by ml^2, which we know is a positive constant. Defining a new constant $a = g/l$ for convenience, this gives us the following function, which will be our candidate Lyapunov function for the system:

$$V(x) = \frac{1}{2}x_2^2 + a(1 - \cos x_1) \tag{1.213}$$

Let's first verify that $V(x)$ is a valid Lyapunov function, and show that it is at least locally positive definite. Let's use our simpler, alternate characterization of locally positive definite functions. By the alternate characterization, we recall that a $V(x)$ is a locally positive definite function if $V(0) = 0$ and $V(x) > 0$ for some region of nonzero x around the origin.
Does these properties hold for our candidate Lyapunov function? First, we check the condition that $V(0) = 0$. Plugging $x_1 = x_2 = 0$ into $V(x)$, we get:

$$V(0) = \frac{1}{2}0^2 + a(1 - \cos 0) = 0 \tag{1.214}$$

Thus, the first condition is satisfied. Does our function satisfy the second condition? We want to show for some values of x that:

$$V(x) = \frac{1}{2}x_2^2 + a(1 - \cos x_1) > 0 \tag{1.215}$$

Firstly, we know that the quadratic term, $\frac{1}{2}x_2^2$, is always greater than zero for all nonzero values of x_2. What about the second term?
Using our knowledge of trigonometry, we know that for all values of x_1, the term $a(1 - \cos x_1)$ is bounded below by 0 and bounded above by $2a$.

$$0 \leq a(1 - \cos x_1) \leq 2a \tag{1.216}$$

Since neither component of $V(x)$ is negative and our overall function is nonzero for some region of x around the origin, we conclude that our candidate Lyapunov function, $V(x)$, is a locally positive definite function.
Next, let's analyze the derivative of the Lyapunov function! Let's start by taking

the derivative of V, and follow up by substituting in the constraint of the system dynamics. Using the chain rule for taking derivatives:

$$\dot{V}(x) = x_2\dot{x}_2 + a\dot{x}_1 \sin x_1 \tag{1.217}$$

$$= x_2\dot{x}_2 + ax_2 \sin x_1 \tag{1.218}$$

Now, we substitute in the constraint of the system dynamics: $\dot{x}_2 = -a \sin x_1 - \beta x_2$ (recall that we defined $a = g/l$).

$$\dot{V}(x) = x_2(-a \sin x_1 - \beta x_2) + ax_2 \sin x_1 \tag{1.219}$$

$$= -\beta x_2^2 \tag{1.220}$$

Since $\beta > 0$, this tells us that $\dot{V} \leq 0$ for all values of x! Thus, by the direct method of Lyapunov, we conclude that the equilibrium point $x_e = [0, 0]$ of the pendulum is locally stable in the sense of Lyapunov.

Note that because the derivative of our Lyapunov function, $\dot{V} = -\beta x_2^2$, *only* depends on x_2, we cannot make any conclusions about whether $-\dot{V}$ is locally positive definite! For example, if we set $x_1 \neq 0$ and $x_2 = 0$, we will have a nonzero x where $\dot{V} = 0$.

Since stronger forms of Lyapunov stability require $-\dot{V}$ to be positive definite or locally positive definite, we see that using this *particular* Lyapunov function, we cannot show that $x_e = [0, 0]$ is asymptotically stable.

As it happens, with a different choice of Lyapunov function, we *could* actually prove that the equilibrium point $x_e = 0$ of the pendulum is asymptotically stable! For this system, the Lyapunov function:

$$V(x) = \frac{1}{2}x^T P x + a(1 - \cos x_1) \tag{1.221}$$

Where $P \in \mathbb{R}^{2\times 2}$ is a positive definite matrix, could be used to show x_1 is actually locally asymptotically stable! The existence of this second Lyapunov function with stronger stability conditions highlights several *extremely* important points about working with Lyapunov functions!

1. A Lyapunov function $V(x)$ is *not* unique! For a system $\dot{x} = f(x, t)$, there can be *any number* of valid Lyapunov functions.
2. Certain Lyapunov functions allow you to conclude stronger types of stability than others! Even if one Lyapunov function tells you a point is just SISL, there might be another Lyapunov function that tells you it's asymptotically stable as well!
3. The stability of an equilibrium point is described by the *strongest* form of stability we are able to prove. If one Lyapunov function says a point is SISL and another says the point is asymptotically stable, we use the stronger condition - that point must be asymptotically stable.

When working with Lyapunov functions, always remember - just because we can't find a Lyapunov function that satisfies the conditions we're looking for *doesn't mean* that one doesn't exist! For many systems, finding the right Lyapunov function is a challenging process.

Stability of Linear Systems

So far, we've dealt strictly with studying the stability of general nonlinear systems of the form:

$$\dot{x} = f(x,t),\ x \in \mathbb{R}^n,\ t \in \mathbb{R} \tag{1.222}$$

In this section, we'll turn our attention to the stability of linear, time invariant systems of the form:

$$\dot{x} = Ax,\ x \in \mathbb{R}^n,\ A \in \mathbb{R}^{n \times n} \tag{1.223}$$

As we saw in the previous section, it can be challenging to determine the stability of general nonlinear systems! For this simpler linear system, what conclusions can we make? As we'll soon discover, linear algebra offers us an exceptionally convenient and elegant array of tools we can use to describe the stability of $\dot{x} = Ax$.

Additionally, as we'll soon learn in our study of the indirect method of Lyapunov, the techniques we develop in linear stability may actually be used to inform our knowledge of *nonlinear* stability.

Let's get started by solving for a set of conditions that guarantee *global exponential stability* of the equilibrium point $x_e = 0$ for the linear system $\dot{x} = Ax$.

We'll start our search for these conditions by reviewing the general solution to linear, time invariant differential equations. Recall that for all matrices $A \in \mathbb{R}^{n \times n}$ and all initial conditions $x_0 \in \mathbb{R}^n$, the solution to the initial value problem:

$$\dot{x} = Ax,\ x(0) = x_0 \tag{1.224}$$

Is given by the following expression:

$$x(t) = e^{At}x_0 \tag{1.225}$$

Where e^{At} is the matrix exponential of A, defined according to the Taylor series of the scalar exponential function. Thus, unlike the general nonlinear case, we actually *know* how to solve every linear, time invariant differential equation! Thus, we can attempt to study the stability of a linear system by using $x(t)$ and *directly* applying the definitions of Lyapunov stability.

Before we get to work on applying this general solution to study stability, we have one more piece of the puzzle to solve! Recall that earlier, when searching for a method to write the matrix exponential in a closed form, we found that if we have a diagonal matrix with n potentially repeated eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$:

$$A = \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.226}$$

We may compute the matrix exponential, $exp(At)$, through the following convenient closed form expression:

$$e^{At} = \begin{bmatrix} e^{\lambda_1 t} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & e^{\lambda_n t} \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{1.227}$$

This result, while immensely useful in the study of differential equations, is *not* something we can apply to every matrix $A \in \mathbb{R}^{n \times n}$! From our knowledge of linear algebra, we know that we *cannot* place every matrix $A \in \mathbb{R}^{n \times n}$ in a diagonal form!

Thus, if we wish to study the stability of a *general* linear system by using $x(t) = e^{At}x_0$, this is not a result we can make use of! We must find *another* way to compute a closed form of $exp(At)$ even when A is not diagonalizable. How can we achieve this?

Let's see if we can transform any matrix into a form that's *close* to a diagonal matrix! The **Jordan canonical form** (JCF), or "Jordan form," of a matrix is arguably the next best thing to diagonalization, and may be applied to *any square matrix*, diagonalizable or not. The key to the applicability of the Jordan canonical form is that it provides us with a reliable way to deal with *repeated eigenvalues*.

How can we compute the Jordan form of a matrix? Suppose we have a matrix $A \in \mathbb{R}^{n \times n}$ with m unique (not repeated) eigenvalues, $\lambda_1, \lambda_2, ..., \lambda_m$. We know that the characteristic polynomial of the matrix can be written:

$$\Delta(\lambda) = (\lambda - \lambda_1)^{m_1}(\lambda - \lambda_2)^{m_2}...(\lambda - \lambda_m)^{m_m} \tag{1.228}$$

Where m_i, called the *algebraic multiplicity* of the eigenvalue λ_i, is the number of times λ_i is repeated. The Jordan form of a matrix with this characteristic polynomial is computed:

$$J = \begin{bmatrix} J_1 & 0 & \dots & 0 \\ 0 & J_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & J_m \end{bmatrix} \in \mathbb{R}^{n \times n}, \; J_i = \begin{bmatrix} \lambda_i & 1 & \dots & 0 \\ 0 & \lambda_i & \ddots & \vdots \\ \vdots & & \ddots & 1 \\ 0 & \dots & 0 & \lambda_i \end{bmatrix} \in \mathbb{R}^{m_i \times m_i}, \tag{1.229}$$

The Jordan form of a matrix is a **block diagonal** matrix, where a set of matrices known as **Jordan blocks** are found along the diagonal of J and zeros are found everywhere else. Each Jordan block, J_i, is an upper triangular matrix with the eigenvalue λ_i on the diagonal, a strip of $1s$ just above the diagonal, and zeros everywhere else. The size of each Jordan block is determined by the algebraic multiplicity of the eigenvalue λ_i.

Suppose, for instance, that the eigenvalue λ_i has an algebraic multiplicity of 3.

The Jordan block associated with that eigenvalue would be:

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 \\ 0 & \lambda_i & 1 \\ 0 & 0 & \lambda_i \end{bmatrix} \in \mathbb{R}^{3\times 3} \tag{1.230}$$

By finding the Jordan block associated with each eigenvalue and placing each block along the diagonal of the matrix J, we can piece together a matrix's Jordan canonical form.

Since we can find a Jordan form for any matrix A, we conclude from our knowledge of linear algebra that for all $A \in \mathbb{R}^{n\times n}$, diagonalizable or not, there exists a transformation matrix $T \in \mathbb{R}^{n\times n}$ such that:

$$A = T^{-1}JT \tag{1.231}$$

Where J is the Jordan form of A. Note that here, we'll just use the Jordan form as a tool in our mathematical analysis, and won't focus on the details of finding the transformation T.[6]

Let's bring our discussion of the Jordan form back around to solving for the matrix exponential of an arbitrary matrix. One of the main advantages of the Jordan form is that the matrix exponential of an arbitrary Jordan form J has a simple closed form expression!

Suppose we have a Jordan form matrix $J \in \mathbb{R}^{n\times n}$ with m Jordan blocks:

$$J = \begin{bmatrix} J_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & J_m \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{1.232}$$

Just like we can compute the matrix exponential of a diagonal matrix by taking the exponential of each diagonal entry, we can compute the matrix exponential of a block diagonal matrix by taking the exponential of each block diagonal entry:

$$e^{Jt} = \begin{bmatrix} e^{J_1 t} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & e^{J_m t} \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{1.233}$$

How can we compute the matrix exponential of each Jordan block? We can show that the matrix exponential of a $p \times p$ Jordan block J_i with an eigenvalue λ_i is computed:

$$e^{J_i t} = \begin{bmatrix} e^{\lambda_i t} & te^{\lambda_i t} & \frac{t^2}{2!}e^{\lambda_i t} & \dots & \frac{t^{p-1}}{(p-1)!}e^{\lambda_i t} \\ 0 & e^{\lambda_i t} & te^{\lambda_i t} & \dots & \frac{t^{p-2}}{(p-2)!}e^{\lambda_i t} \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & \dots & e^{\lambda_i t} \end{bmatrix} \in \mathbb{R}^{p\times p} \tag{1.234}$$

[6] You can consult *Linear Algebra* by Friedberg, Insel, and Spence for a comprehensive treatment of finding such transformations.

For instance, the matrix exponential of a 4×4 Jordan block J_i with eigenvalue λ is computed:

$$e^{J_i t} = \begin{bmatrix} e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \frac{t^3}{3!}e^{\lambda t} \\ 0 & e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} \\ 0 & 0 & e^{\lambda t} & te^{\lambda t} \\ 0 & 0 & 0 & e^{\lambda t} \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{1.235}$$

Although not as clean as the matrix exponential of a *purely* diagonal matrix, the Jordan form offers us a close second and allows us to develop many results in the study of linear differential equations in full generality.
We may apply the Jordan form to prove one of the most fundamental theorems in the study of dynamical systems, regarding the stability of linear systems. Before we study this theorem in full generality, we'll need one more result, which we'll now prove. This result will help us form a tight upper bound on the magnitudes of solutions given by the Jordan form!

Lemma 1 ***Polynomials are Bounded by Exponentials***
If $p(t)$ is an arbitrary real-valued polynomial of degree n, written:

$$p(t) = a_n t^n + a_{n-1}t^{n-1} + ... + a_1 t + a_0,\ a_i \in \mathbb{R} \tag{1.236}$$

For all $\varepsilon > 0$ there exists a constant m such that for all $t \in \mathbb{R}$:

$$p(t) \leq me^{\varepsilon t} \tag{1.237}$$

Proof: We can prove this theorem by strong induction on n, the degree of the polynomial. Recall that a proof by strong induction involves proving a base case, where $n = 0$, and then an inductive case, where we assume the result is true for all integers up to and including n and show that it is true for $n + 1$.
Base Case: ($n = 0$) First, we can prove that this result is true for $n = 0$. In this case, the polynomial will be degree zero, and of the form:

$$p(t) = a_0,\ a_0 \in \mathbb{R} \tag{1.238}$$

For every $\varepsilon > 0$, we can clearly bound this function above by the following:

$$p(t) \leq |a_0|e^{\varepsilon t} \tag{1.239}$$

Since $p(t)$ is a constant function and $e^{\varepsilon t} = 1$ when $t = 0$. Taking $|a_0|$ to be m, this completes the proof of the base case! You can also verify the $n = 1$ case using a little bit of calculus.
Inductive Case: Now, we assume for strong induction that the result is true for all integers between and including 0 and n. Can we show that it's true for $n + 1$? If this is true for all integers between 0 and n, then for a degree n $p(t)$:

$$p(t) = a_n t^n + a_{n-1}t^{n-1} + ... + a_1 t + a_0,\ a_i \in \mathbb{R} \tag{1.240}$$

We can find an $m \in \mathbb{R}$ such that for all $\varepsilon > 0$ and for all t:

$$p(t) \leq me^{\varepsilon t} \tag{1.241}$$

Now, we wish to show that we can find an $m \in \mathbb{R}$ for $n+1$. Let $f(t)$ be an $n+1$ degree polynomial, and $p(t)$ be the n degree polynomial above. Using the inductive hypothesis, we can find an $m_3 \in \mathbb{R}$ such that:

$$f(t) = a_{n+1}t^{n+1} + p(t) \leq a_{n+1}t^{n+1} + m_3e^{\varepsilon t} \tag{1.242}$$

For all $\varepsilon > 0$. Now, if we split up $a_{n+1}t^{n+1} = a_{n+1}t^n t$, we realize that $a_{n+1}t^{n+1}$ is the product of a degree n polynomial and a degree 1 polynomial! Thus, for all $\varepsilon > 0$, there exist m_1, m_2, m_3 such that:

$$\begin{aligned} f(t) &\leq a_{n+1}t^{n+1} + m_3e^{\varepsilon t} && (1.243)\\ &= a_{n+1}(t^n)(t) + m_3e^{\varepsilon t} && (1.244)\\ &\leq a_{n+1}(m_1e^{\frac{\varepsilon}{2}t})(m_2e^{\frac{\varepsilon}{2}t}) + m_3e^{\varepsilon t} && (1.245)\\ &= a_{n+1}m_1m_2e^{\varepsilon t} + m_3e^{\varepsilon t} && (1.246)\\ &= (a_{n+1}m_1m_2 + m_3)e^{\varepsilon t} && (1.247) \end{aligned}$$

Note that in the above, we bounded $a_{n+1}t^n$ and t above by exponents with rates $\varepsilon/2$ instead of just ε! We were able to do this since the exponential bounding condition holds *for all* values of ε, so it didn't matter if we used ε or $\varepsilon/2$. This choice allowed us to factor out an $exp(\varepsilon t)$ later down the line!
Defining $m = a_{n+1}m_1m_2 + m_3$, we have shown that for an $n+1$ degree polynomial, $f(t)$, there exists an m such that:

$$f(t) \leq me^{\varepsilon t} \tag{1.248}$$

For all $\varepsilon > 0$. This completes the proof of the inductive case! Thus, by induction, this result holds for arbitrary polynomials. This completes the proof. □

With this lemma in our toolkit, we're now ready to solve for the conditions of general stability of linear time invariant systems! The following proposition is one we'll take with us across our study of feedback control.
As with our lemma about exponential functions, to prove results about the stability of LTI systems in the most general case, we will need to use some results from real analysis. The proof of the following theorem is quite challenging, but can be good practice for building intuition on forming bounds and applying the definitions of stability. You're encouraged to give it a try!

Proposition 3 ***Exponential Stability of LTI Systems***
The equilibrium point $x_e = 0$ of the linear, time invariant system:

$$\dot{x} = Ax, x \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n} \tag{1.249}$$

Is globally exponentially stable if and only if $Re(\lambda_i) < 0, 1 \leq i \leq n$, where λ_i are the eigenvalues of A.

This theorem states that the real components of the eigenvalues of A completely characterize the exponential stability of the system! Thus, without needing to calculate a Lyapunov function, we're able to conclude the strongest form of stability for a linear system just by solving for the eigenvalues of A!

Proof: First, we'll proceed by showing that if all of the eigenvalues of A have negative real components, $x_e = 0$ is a globally stable equilibrium point. Recall that formally, $x_e = 0$ is a globally exponentially stable equilibrium point of a system if there exists an $\alpha > 0$ and an m such that for all $x(0) = x_0 \in \mathbb{R}^n$:

$$||x(t)|| \leq me^{-\alpha t}||x_0|| \tag{1.250}$$

We know that the solution $x(t)$ to the system is given by:

$$x(t) = e^{At}x_0 \tag{1.251}$$

Let's use the Jordan form to compute the matrix exponential of A! Since every matrix has a Jordan form, we know there exists a transformation T such that $A = T^{-1}JT$. Defining a change of variables $z = Tx$, we may rewrite the solution in our new coordinates:

$$z(t) = e^{Jt}z_0 \tag{1.252}$$

We know that e^{Jt} is computed:

$$e^{Jt} = \begin{bmatrix} e^{J_1 t} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & e^{J_m t} \end{bmatrix}, \; e^{J_i t} = \begin{bmatrix} e^{\lambda_i t} & \dots & \frac{t^{p-1}}{(p-1)!}e^{\lambda_i t} \\ \vdots & \ddots & \vdots \\ 0 & \dots & e^{\lambda_i t} \end{bmatrix} \in \mathbb{R}^{p \times p} \tag{1.253}$$

Where p is the algebraic multiplicity of the eigenvalue λ_i. Here's our general strategy for this proof. If we can show that the section of the vector $z(t)$ associated with *each* Jordan block is exponentially bounded, we can show that the magnitude of the *entire* vector is exponentially bounded!
Let's name the portion of $z(t)$ associated with the Jordan block J_i $z_i'(t)$. What can we conclude about $||z_i'(t)||$? Since J is block diagonal, we know:

$$z_i'(t) = e^{J_i t}z_{0i}' \tag{1.254}$$

Where z_{0i}' is the portion of the initial condition associated with the i^{th} Jordan

block. Multiplying out $exp(J_i t)$ and z'_{0i} and taking the norm, we get:

$$||e^{J_i t} z'_0|| = || \begin{bmatrix} e^{\lambda_i t} z'_{0i1} + ... + \frac{t^{p-1}}{(p-1)!} e^{\lambda_i t} z'_{0ip} \\ \vdots \\ e^{\lambda_i t} z'_{0ip} \end{bmatrix} || \tag{1.255}$$

$$= ||e^{\lambda_i t} \begin{bmatrix} z'_{0i1} + ... + \frac{t^{p-1}}{(p-1)!} z'_{0ip} \\ \vdots \\ z'_{0ip} \end{bmatrix} || \tag{1.256}$$

Now, we'd like to start forming some inequalities to bound this norm! Let's define z_{max} as the maximum magnitude of the initial condition entries:

$$z_{max} = \max\{|z'_{0i1}|, |z'_{0i2}|, ..., |z'_{0ip}|\} \tag{1.257}$$

Using z_{max}, we can form the following inequality:

$$||e^{J_i t} z'_{0i}|| \leq ||e^{\lambda_i t} \begin{bmatrix} z_{max} + ... + \frac{t^{p-1}}{(p-1)!} z_{max} \\ \vdots \\ z_{max} \end{bmatrix} || \tag{1.258}$$

$$= ||e^{\lambda_i t} z_{max} \begin{bmatrix} 1 + ... + \frac{t^{p-1}}{(p-1)!} \\ \vdots \\ 1 \end{bmatrix} || \tag{1.259}$$

Now, in each entry of the vector, we have a polynomial with positive coefficients! Applying the lemma we discussed above, we know that if $p(t)$ is an arbitrary polynomial, there exists a constant m such that for all $\varepsilon > 0$ and all $t \in \mathbb{R}^+$:

$$p(t) \leq m e^{\varepsilon t} \tag{1.260}$$

Thus, building off of our previous step, where we factored out z_{max}, we may make another inequality by bounding each polynomial by an exponential:

$$||e^{J_i t} z'_{0i}|| \leq ||e^{\lambda_i t} z_{max} \begin{bmatrix} m e^{\varepsilon t} \\ \vdots \\ m e^{\varepsilon t} \end{bmatrix} || \tag{1.261}$$

$$= ||e^{\lambda_i t} z_{max} m e^{\varepsilon t} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} || = |e^{\lambda_i t}| z_{max} m e^{\varepsilon t} || \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} || \tag{1.262}$$

Note that we take the absolute value of $exp(\lambda_i t)$ when removing it from the norm because λ_i has the potential to be complex! Let's simplify the expression above, and name the norm of the vector of ones l. If we define $m' = z_{max} m l$, this tells us that for all values of $\varepsilon > 0$:

$$||z'_i(t)|| = ||e^{J_i t} z'_{0i}|| \leq m' |e^{\lambda_i t}| e^{\varepsilon t} \tag{1.263}$$

With this inequality, we've almost reached the result that we want for each Jordan block! Our next step is to find a way to drop the absolute value of the exponential in the middle of this expression. How can we do this?
If λ_i is complex, then for real $a, b \in \mathbb{R}$, we can express λ_i as:

$$\lambda_i = a + bi \in \mathbb{C} \tag{1.264}$$

Let's express the exponential $e^{\lambda_i t}$ in terms of a and b!

$$e^{\lambda_i t} = e^{(a+bi)t} = e^{at+bit} = e^{at}e^{bit} \tag{1.265}$$

How can we compute the exponential of bit? We may apply Euler's formula, which says:

$$e^{bit} = \cos bt + i \sin bt \tag{1.266}$$

What conclusions can we make about the magnitude of the term $exp(bit)$? Using the definition of the magnitude of a complex number, we get:

$$|e^{bit}| = \sqrt{\cos^2 bt + \sin^2 bt} = 1 \tag{1.267}$$

Let's use this knowledge to find an expression for $|exp(\lambda_i t)|$!

$$|e^{\lambda_i t}| = |e^{at}||e^{bit}| = e^{at} \tag{1.268}$$

Thus, the value of $|e^{\lambda_i t}|$ is *entirely* determined by the real component of λ_i! Going back to our bound for $z_i'(t)$, we therefore conclude that:

$$||z_i'(t)|| \leq m'|e^{\lambda_i t}|e^{\varepsilon t} = m'e^{at}e^{\varepsilon t} = m'e^{(a+\varepsilon)t} \tag{1.269}$$

This completes the our study of the Jordan block J_i! Let's bring this result back to the full system! We know that the general solution to the differential equation $\dot{z} = Jz$ is given by:

$$z(t) = e^{Jt}z_0 = \begin{bmatrix} e^{J_1 t} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & e^{J_m t} \end{bmatrix} \begin{bmatrix} z_{01}' \\ \vdots \\ z_{0m}' \end{bmatrix} = \begin{bmatrix} e^{J_1 t}z_{01}' \\ \vdots \\ e^{J_m t}z_{0m}' \end{bmatrix} \tag{1.270}$$

Where z_{0i}' is once again the portion of the initial condition vector corresponding to the i^{th} Jordan block. Let's split up $z(t)$ into a set of m vectors corresponding to *each* Jordan block!

$$z(t) = \begin{bmatrix} e^{J_1 t}z_{01}' \\ \vdots \\ e^{J_m t}z_{0m}' \end{bmatrix} = \begin{bmatrix} e^{J_1 t}z_{01}' \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ e^{J_2 t}z_{02}' \\ \vdots \\ 0 \end{bmatrix} + ... + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ e^{J_m t}z_{0m}' \end{bmatrix} \tag{1.271}$$

From the triangle inequality, we know that for any two vectors u, v, it is true that $||u+v|| \leq ||u|| + ||v||$! Applying this fact to the m vectors above, we have:

$$||z(t)|| \leq || \begin{bmatrix} e^{J_1 t} z'_{01} \\ 0 \\ \vdots \\ 0 \end{bmatrix} || + || \begin{bmatrix} 0 \\ e^{J_2 t} z'_{02} \\ \vdots \\ 0 \end{bmatrix} || + ... + || \begin{bmatrix} 0 \\ \vdots \\ 0 \\ e^{J_m t} z'_{0m} \end{bmatrix} || \tag{1.272}$$

$$= ||e^{J_1 t} z'_{01}|| + ||e^{J_2 t} z'_{02}|| + ... + ||e^{J_m t} z'_{0m}|| \tag{1.273}$$

Now, we can apply the bound that we derived above for each Jordan block! We know that for all Jordan blocks, $||e^{J_i t} z'_{0i}|| \leq m_i exp((a_i + \varepsilon)t)$ for all $\varepsilon > 0$, where a_i is the real component of the i^{th} eigenvalue, λ_i.
Since each of the vectors above is simply $e^{J_i t} z'_{0i}$ along with a set of zeros, we may apply the individual Jordan block bounds to arrive at the following inequality:

$$||z(t)|| \leq ||e^{J_1 t} z'_{01}|| + ||e^{J_2 t} z'_{02}|| + ... + ||e^{J_m t} z'_{0m}|| \tag{1.274}$$

$$\leq m_1 e^{(a_1+\varepsilon)t} + m_2 e^{(a_2+\varepsilon)t} + ... + m_m e^{(a_m+\varepsilon)t} \tag{1.275}$$

Which holds for all $\varepsilon > 0$! We're almost there! We'd now like to condense this inequality into a single exponential! To do this, we pick $m' = \max\{m_1, m_2, ..., m_m\}$. Using this m', we conclude:

$$||z(t)|| \leq m_1 e^{(a_1+\varepsilon)t} + m_2 e^{(a_2+\varepsilon)t} + ... + m_m e^{(a_m+\varepsilon)t} \tag{1.276}$$

$$\leq m'(e^{(a_1+\varepsilon)t} + e^{(a_2+\varepsilon)t} + ... + e^{(a_m+\varepsilon)t}) \tag{1.277}$$

This brings us one step closer to where we need to be! If we look at the expression above, we have exponentials with m potentially different real parts of eigenvalues! How can we condense all of these exponentials into a single exponential? Let's pick:

$$a = \max\{a_1, a_2, ..., a_m\} \tag{1.278}$$

Using this a, we form another inequality:

$$||z(t)|| \leq m'(e^{(a_1+\varepsilon)t} + e^{(a_2+\varepsilon)t} + ... + e^{(a_m+\varepsilon)t}) \tag{1.279}$$

$$\leq m'(e^{(a+\varepsilon)t} + e^{(a+\varepsilon)t} + ... + e^{(a+\varepsilon)t}) \tag{1.280}$$

$$= m' m e^{(a+\varepsilon)t} \tag{1.281}$$

This inequality hold for all values of $\varepsilon > 0$. Now comes one of the most important steps of the proof! If the real parts of *all* of the eigenvalues of J are less than zero, the maximum of these values, a, will *also* be less than zero! Since ε can be *any* positive value, we can pick ε such that:

$$a + \varepsilon < 0 \tag{1.282}$$

Let's define a new constant $\alpha = -(a + \varepsilon) > 0$. Rewriting the inequality above using this α:

$$||z(t)|| \leq m'me^{-\alpha t} \tag{1.283}$$

Thus, $z(t)$ is bounded above by a decaying exponential! At this stage, we're almost finished with the proof - all that remains is to transform $z(t)$ back into our original coordinates, $x(t)$. We know from our original transformation that:

$$||x(t)|| = ||T^{-1}z(t)|| \tag{1.284}$$

How can we bound the value of $||T^{-1}z(t)||$ in terms of the value of $||z(t)||$? We can use the matrix 2-norm of T^{-1}, which is equivalent to the maximum singular value of T^{-1}. By definition of the matrix norm, for all z:

$$||T^{-1}z|| \leq \sigma_{max}||z|| \tag{1.285}$$

Where σ_{max} is the maximum singular value of T^{-1}. Applying this to the above, we have:

$$||x(t)|| = ||T^{-1}z(t)|| \leq \sigma_{max}||z(t)|| \leq \sigma_{max}m'me^{-\alpha t} \tag{1.286}$$

All that remains is to define a constant that allows us to match this inequality perfectly with the definition of exponential stability. Assuming nonzero $||x_0||$ (the zero case is automatically bounded above by an exponential with any scale factor), we pick:

$$L = \frac{\sigma_{max}m'm}{||x_0||} \tag{1.287}$$

Using this L, we have that:

$$||x(t)|| \leq \frac{\sigma_{max}m'm}{||x_0||}e^{-\alpha t}||x_0|| \tag{1.288}$$

$$||x(t)|| \leq Le^{-\alpha t}||x_0|| \tag{1.289}$$

Where $\alpha > 0$ as long as all eigenvalues of A have negative real components. This bound perfectly matches the definition of exponential stability, and holds for all initial conditions x_0. Thus, we conclude that if the eigenvalues of A have all negative real components, the system:

$$\dot{x} = Ax,\ x(0) = x_0 \tag{1.290}$$

Is globally exponentially stable. The *only if* portion of this theorem can be shown to be satisfied by the uniqueness of the solution to the linear differential equation. This completes the proof! Phew, that was a lot of inequalities![7] □

[7] This theorem may also be proven by looking at different matrix norms of e^{At} - we used a slightly different approach to avoid introducing too many new concepts.

The Indirect Method of Lyapunov

What other methods are there for analyzing the stability of nonlinear systems? As we saw in the previous sections, finding the right Lyapunov function to analyze the dynamics of a system can be a challenging process! Are there any cases where we can avoid the need for a Lyapunov function?
Recall that in our earlier study of dynamical systems, we used the method of Jacobian linearization to locally approximate nonlinear systems as linear ones! Now that we have a set of tools for analyzing the stability of linear systems, could we perhaps use our local, linear approximation to make conclusions about the stability of the original nonlinear system? This question leads us to the indirect method of Lyapunov.

Theorem 4 ***Stability by Linearization (Indirect Method of Lyapunov)*** *Suppose the system $\dot{x} = f(x,t)$ has an equilibrium point $x_e = 0$, and a Jacobian linearization about $x_e = 0$ of $\dot{x} \approx A(t)x$. If $A(t)$ is a bounded matrix, and the difference between the linearization and the original dynamics satisfy:*

$$\lim_{||x||\to 0} \sup_{t\geq 0} \frac{||f(x,t) - A(t)x||}{||x||} = 0 \tag{1.291}$$

We may conclude the stability of $x_e = 0$ as follows. If $x_e = 0$ is a uniformly asymptotically stable equilibrium point of $\dot{x} = A(t)x$, then $x_e = 0$ is a locally uniformly asymptotically stable equilibrium point of $\dot{x} = f(x,t)$.

What does the condition $\lim_{||x||\to 0} \sup_{t\geq 0}(...) = 0$ mean? This means that as we bring x closer to the equilibrium point, the largest possible difference between $f(x,t)$ and the approximation $A(t)x$ as we vary time across all of its possible values should approach zero faster than $||x|| \to 0$.
Thus, for this theorem to hold, we require that the Jacobian linearization will closely approximate $f(x,t)$ near the equilibrium point for all time. This type of convergence between $f(x,t)$ and $A(t)x$ is a special form of convergence known as *uniform convergence*.
Let's summarize what this theorem is saying. This theorem states the if the Jacobian linearization of a system about $x_e = 0$ exists and is well-behaved, we can conclude local stability of the *original* nonlinear system just by looking at the stability of its Jacobian linearization! Thus, by finding the eigenvalues of the Jacobian linearization and ensuring their real components are all less than zero, we can conclude local asymptotic stability of $\dot{x} = f(x,u)$!
Why does this result only hold locally? Recall that the Jacobian linearization is only a good local approximation of a nonlinear system - once we stray far away from the equilibrium point, the error between the linear approximation and the system may grow at an unbounded rate.
Because of this, we can only use the linearization to study *local* stability. To

study global stability, we generally still must use the direct method of Lyapunov, and find a valid Lyapunov function. Interested readers are encouraged to consult *A Mathematical Introduction to Robot Manipulation* by Murray, Li, and Sastry, or *Feedback Systems* by Murray for a brief description of LaSalle invariance, another technique used in nonlinear stability analysis.

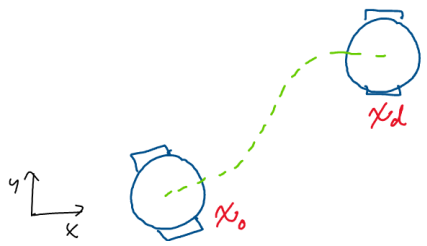
Chapter 2

Feedback Control Fundamentals

Thus far, we've developed a significant amount of theory in the study of nonlinear systems. We've gone from developing a set of conventions for describing nonlinear differential equations to describing the solutions to linear systems through to an extensive study of stability.
In this chapter, we'll put these fundamental concepts to work, and study some key ideas in feedback control! We'll discuss important aspects of control theory from linear to nonlinear control design, and will build a strong foundation for more advanced techniques.

2.1 Motivating Feedback

Let's begin our development of control theory by defining and motivating the value of *feedback* in the control of robotic systems. Once we've sufficiently investigated the *what* and *why* concepts of control, we'll move on to the *how*, and learn about some foundational techniques in control.
Firstly, what *is* control theory? The field of control theory is largely concerned with finding the best possible input to a dynamical system to get the system to behave how we want. For instance, in robotics, we might want to drive a turtlebot from some starting position, x_0, to some desired position, x_d.



Above: How can we drive a turtlebot from x_0 to follow $x_d(t)$?

How can we find the optimal input to the system that allows us to move from position x_0 to x_d quickly and precisely? Can we prove rigorously that the system will converge to our desired state, and if so, can we find how quickly it will converge? These are some of the fundamental questions we'll ask in this chapter!

Let's perform some expository analysis of these questions, and set up a small example problem. Along the way, we'll identify some common pitfalls in control design, and gain some perspective for the task ahead of us!

Imagine that we want to control our turtlebot, which has dynamics described by a nonlinear differential equation:

$$\dot{x} = f(x, u),\ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{2.1}$$

We recall that we can control x, the state vector, by choosing u, the input vector to the system. Remember that the input vector, u, is something that's entirely under our control!

Let's see if we can find an expression for an input u that allows our turtlebot to track a desired trajectory that changes with time, $x_d(t)$. One approach to solving this problem is to use the system dynamics, which tell us the relationship between the input u and the state vector, x! Starting with the differential equation $\dot{x} = f(x, u)$, we have:

$$\frac{dx}{dt} = f(x, u) \tag{2.2}$$

$$dx = f(x, u)dt \tag{2.3}$$

We may now integrate the left side from time 0 to time t, and the right side from an initial state $x(0) = x_0$ to the desired state at time t, $x_d(t)$. Using a dummy variable τ in the place of t to maintain proper mathematical notation:

$$\int_{x_0}^{x_d(t)} dx = \int_0^t f(x(\tau), u(\tau))d\tau \tag{2.4}$$

$$x_d(t) - x_0 = \int_0^t f(x(\tau), u(\tau))d\tau \tag{2.5}$$

$$x_d(t) = x_0 + \int_0^t f(x(\tau), u(\tau))d\tau \tag{2.6}$$

If we could somehow evaluate the integral on the right hand side and solve for $u(t)$ in terms of x_d and the other system variables, we would be able to find an input $u(t)$ that allows our turtlebot to track the desired trajectory $x_d(t)$. Although this might seem to be successful choice of input at an initial glance, this strategy actually breaks down in several places!

This strategy of solving for $u(t)$ is known as an **open loop** control law. Open loop controllers don't use any information about the *actual* state of the vehicle, and simply rely on a knowledge of system dynamics and desired trajectory to

decide the input to a system.
In our example above, for instance, we only used the turtlebot dynamics, $\dot{x} = f(x, u)$, and the desired trajectory $x_d(t)$, to compute a control input! We didn't rely on any information from sensors onboard the turtlebot, which would tell us the actual state of the vehicle.

Above: An open loop controller reads the desired state and sends an input to the system, which moves the system to a state x.

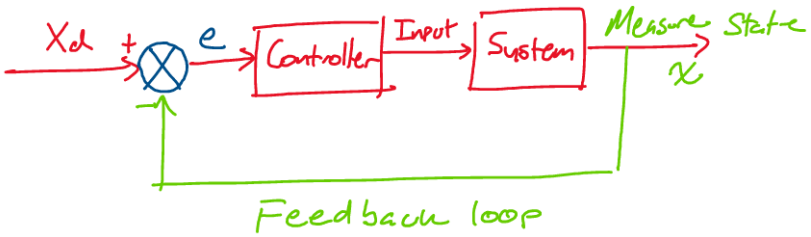
Why might this be undesirable? If we were to use this choice of input on actual hardware, we would encounter several problems. Firstly, the model of the system dynamics, one of the main things we rely on in our open loop input, never translates perfectly to the real world!
In a real physical environment, there will always be effects we haven't modeled. To name a few for the turtlebot, forces such as friction, rolling resistance, and motor temperature could all impact how the turtlebot really moves. Furthermore, there is the potential for unknown, unexpected disturbances to throw off the model! For example, if someone were to bump into the turtlebot while it moves, our original dynamics equation, $\dot{x} = f(x, u)$, would have no way of accounting for this!
This means that if we were to run this open loop controller on a real robot, even though our control input might work perfectly in an ideal environment, the real robot will quickly move away from the desired trajectory!
If the robot doesn't use any information about its *actual* state from sensors, and assumes its theoretical dynamics model $\dot{x} = f(x, u)$ is perfect, it has no way to correct for any unexpected environmental disturbances that may occur.

Above: An open loop controller will quickly diverge from the desired path in the presence of unknown disturbances.

How can we stop this kind of divergence from the desired trajectory from happening and make our controller *robust* to unexpected environmental disturbances? Instead of solely relying on our dynamics model, we can incorporate *feedback* from sensors that tell us where our robot actually is at all times!

In the case of the turtlebot, for instance, by using sensors that tell us the real-world position and speed of the vehicle, we'll be able to better find an input that allows us to correct for things like environmental disturbances! If an unexpected force pushes the turtlebot off of its path, the sensors will recognize this, and adjust the input to the system to put it back on the right track.

Closed loop feedback controllers are the class of controllers that determine the input to a system by incorporating real-world information from sensors. We can visualize a simple closed loop feedback controller using the following diagram:



Above: A simple feedback control system.

Let's break down the different components of this diagram. First, a desired state, x_d, is passed into the system. Next, the desired state is compared to the actual measured state of the system, which is determined by sensors. The difference between the desired and measured states is passed into a feedback controller, which uses the difference to determine an input, u. This input is then passed into the system and the resulting state is measured by the sensors. The process then continues!

Notice how in the diagram above, the measured state is *fed back* to the start of the diagram, where it is compared to the desired state. This is what gives feedback control its name.

In the study of control, we want to develop feedback controllers that give us certain guarantees on performance, stability, and safety. In the following sections, we'll learn the fundamentals of controller design and analysis, and see how an appropriate choice of u can be determined!

2.2 Linear Control

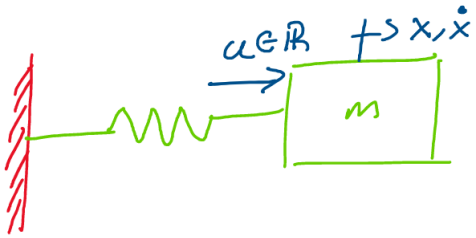
We'll begin our discussion of feedback control by focusing on the class of linear, time invariant systems. Recall that these systems have the form:

$$\dot{x} = Ax + Bu,\ A \in \mathbb{R}^{n\times n},\ B \in \mathbb{R}^{n\times m}, x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{2.7}$$

$$y = Cx + Du,\ C \in \mathbb{R}^{p\times n},\ D \in \mathbb{R}^{p\times m}, y \in \mathbb{R}^p \tag{2.8}$$

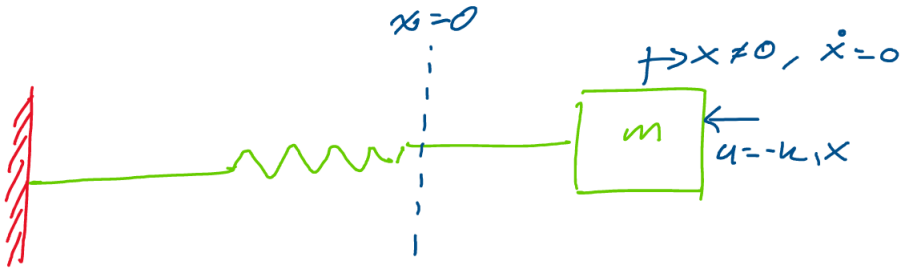
Generally, we can split control design problems into two classes: stabilization (sometimes called regulation) and tracking. In a stabilization problem, we wish to find an input that *stabilizes* the system around one of its equilibrium points. In a tracking problem, on the other hand, we want to find a control input that

allows the system to follow some desired trajectory, $x_d(t)$.
Let's start by tackling the simpler problem, stabilization! How can we design a controller to stably bring the system to an equilibrium point from any initial condition, potentially in the presence of disturbances?
Let's try and reason about this question by examining a simple system: a mass that oscillates on a spring with a force input, u. Recall that the only equilibrium point of this system is the point $[x, \dot{x}] = [0, 0]$, where the spring is unstretched and the mass has zero velocity.



Above: How can we stabilize the mass to its equilibrium point?

Let's consider a couple of different states of the system, and see if we can use our intuition to come up with an input that will drive the system back to its equilibrium in each case! First, consider the case where the mass is too far to the right of the equilibrium point, but has zero velocity.

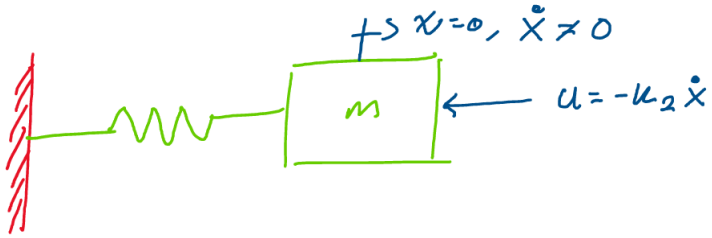


Above: How can we drive the system back to $x = 0$?

In this case, we want to drive the system back to the left, towards the point $x_e = [0, 0]$. To achieve this, we can apply an input to the system in the negative x direction.
How strong should this input be? Intuitively, the further away we are from the equilibrium point $x = 0$, the stronger our input should be, and the closer we are to the equilibrium point, the smaller our input should be. Thus, when we're at this state, we can use an input that points in the negative x direction and is proportional to the distance to the equilibrium point.

$$u = -k_1 x, \; k_1 > 0 \tag{2.9}$$

Let's consider a second case! Now, suppose the string is unstretched ($x = 0$) but the mass is travelling to the right with a high velocity ($\dot{x} > 0$). How can we slow the system down and ensure it returns to $[x, \dot{x}] = [0, 0]$?



Above: How can we slow the mass down and return it to its equilibrium point?

To slow the system down, we could apply our force input opposite to the velocity! As with position, we can scale the magnitude of our input to be proportional to the mass's velocity. The higher the velocity is, the stronger our input should be. With this in mind, we choose the input:

$$u = -k_2 \dot{x}, \; k_2 > 0 \tag{2.10}$$

How could we stabilize the system in the combined case, where *both* x and $\dot{x}$ are nonzero? A natural choice of input would be the *sum* of the two inputs we determined above!

$$u = -k_1 x - k_2 \dot{x}, \; k_1, k_2 > 0 \tag{2.11}$$

As we'll soon see, we can actually prove mathematically that this intuitive choice of input can return the system to its equilibrium point exponentially quickly from *any* initial condition.

Let's generalize our discussion from the simple mass-spring system to the more general linear, time invariant system:

$$\dot{x} = Ax + Bu, \; A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^n, x \in \mathbb{R}^n, u \in \mathbb{R} \tag{2.12}$$

For now, we'll only analyze systems with a single scalar input, u, to simplify our discussion. Suppose that we want to find an input u to *stabilize* this system about its equilibrium point for any initial condition x_0. How can we do this? Inspired by our mass-spring discussion from above, let's try the input:

$$u = -k_1 x_1 - k_2 x_2 - ... - k_n x_n, \; k_i \in \mathbb{R} \tag{2.13}$$

Where each $x_i \in \mathbb{R}$ is an element of the state vector, x. Placing each k_i in a row vector, K, we can rewrite this input as the product of K and the state vector:

$$u = -\begin{bmatrix} k_1 & k_2 & \dots & k_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.14}$$

$$u = -Kx \tag{2.15}$$

In the field of control, K is commonly referred to as a **gain matrix**, and the elements inside K are referred to as **gains**. This choice of input, where we

choose u to be a linear combination of the elements of the state vector, is called **state feedback**. State feedback controllers are extremely popular in linear control, and may be used in many stabilization tasks!
For many linear systems, by correctly choosing the gain matrix, K, we can drive the system to its equilibrium point exponentially quickly from any initial condition.
Note that state feedback is an idea that can be generalized to a multi-input linear system. In the multi-input case, instead of being a row vector, K will be an $m \times n$ matrix.

$$K = \begin{bmatrix} k_{11} & \dots & k_{1n} \\ \vdots & \ddots & \vdots \\ k_{m1} & \dots & k_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \text{ for } u \in \mathbb{R}^m, x \in \mathbb{R}^n \tag{2.16}$$

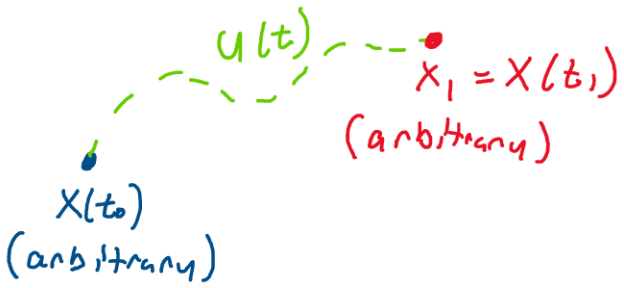
2.2.1 Controllability

Although state feedback will work for a wide variety of A and B matrices, it won't always allow us to stabilize our system! Our ability to stabilize a linear system from any initial condition is closely tied to an idea known as **controllability**. Let's perform a brief study of this idea, and then return to our discussion of state feedback control.

Definition 23 ***Linear Controllability***
A linear system $\dot{x} = Ax + Bu$ is said to be controllable if for all initial conditions $x_0 \in \mathbb{R}^n$ and all $x_1 \in \mathbb{R}^n$, there exists a finite time t_1 and an input $u(t)$ such that if the system starts at x_0 and has the input $u(t)$ applied:

$$x(t_1) = x_1 \tag{2.17}$$

Let's break down what this definition is saying. If a linear system is controllable, we can always find an input $u(t)$ to transport us from any starting state, x_0, to any ending state, x_1, in a finite amount of time!



Above: A linear system is controllable if we can find an input to transport us between any two states in finite time.

If a system is *not* controllable, it's possible that we won't be able to move it to the state we desire! In the case of our stabilization problem from above, for example, if the system $\dot{x} = Ax + Bu$ is *not* controllable, it's possible that there is no control input that will allow us to stabilize our system!
How can we check if a linear system is controllable? The following classic result from control theory gives us a method of verifying the controllability of arbitrary linear, time invariant systems of the form $\dot{x} = Ax + Bu$.

Proposition 4 ***Linear Controllability Matrix***
The linear, time invariant system $\dot{x} = Ax + Bu,\ A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}$ *is controllable if and only if the matrix:*

$$P = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{2.18}$$

Is of rank n*.* P *is known as the controllability matrix of the system.*

This result, which may be proven with a famous result in linear algebra known as the Cayley Hamilton theorem,[1] will be immensely useful in our study of linear control systems.
Based on the definition of controllability, we hypothesize that if a linear system is controllable, and may be driven between any two states in finite time, we can stabilize the system from any initial condition using state feedback, $u = -Kx$. Let's see if we can verify this hypothesis mathematically!

2.2.2 Stabilization

Now that we're equipped with the concept of controllability, let's see if we can prove that controllable linear systems can be stabilized by state feedback! To simplify our analysis, we'll once again consider single input linear systems of the form:

$$\dot{x} = Ax + Bu,\ A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^n, x \in \mathbb{R}^n, u \in \mathbb{R} \tag{2.19}$$

We'll find it convenient to transform our linear system into **controllable canonical form**, a form more convenient for feedback control analysis. Let's briefly discuss the details of this transformation!

Lemma 2 ***Transformation into Controllable Canonical Form***
If the single input linear system $\dot{x} = Ax + Bu,\ x \in \mathbb{R}^n, u \in \mathbb{R}$ *is controllable,*

[1] If you're interested in a proof of this theorem, you're encouraged to consult a text in linear systems theory such as *Linear Systems* by Antsaklis.

there exists a transformation matrix Q that transforms the system into controllable canonical form:

$$\dot{x}_c = A_c x_c + B_c u \tag{2.20}$$

Where $x = Qx_c$ and $A_c = Q^{-1}AQ$, $B_c = Q^{-1}Bu$ are of the form:

$$A_c = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & \dots & -a_{n-1} \end{bmatrix}, \; B_c = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{2.21}$$

The controllable canonical form representation of a linear system will provide us with a much simpler system representation to perform our analysis with. Let's briefly review an important result from linear algebra that will allow us to make use of the controllable canonical form!

Lemma 3 ***Similarity Transformations Preserve Eigenvalues***
Let $A, B \in \mathbb{R}^{n \times n}$. If:

$$B = Q^{-1}AQ \tag{2.22}$$

For some invertible transformation matrix $Q \in \mathbb{R}^{n \times n}$, then A and B are said to be similar matrices, and $Q^{-1}AQ$ is said to be a similarity transformation of A. If A and B are similar, they have the same eigenvalues.

Proof: Let's begin by examining the eigenvalues of A. Suppose λ is an eigenvalue of A with an associated eigenvector $v \in \mathbb{R}^n$. By the definition of an eigenvector:

$$Av = \lambda v \tag{2.23}$$

Multiplying both sides by Q^{-1} and bringing λ to the front:

$$Q^{-1}Av = \lambda Q^{-1}v \tag{2.24}$$

Let's define a new vector $z = Q^{-1}v$. Using z:

$$Q^{-1}AQz = \lambda Q^{-1}Qz \tag{2.25}$$

$$Q^{-1}AQz = \lambda z \tag{2.26}$$

Now, we conclude that if λ is an eigenvalue of A, λ will be an eigenvalue of $Q^{-1}AQ$ as well, with an eigenvector $z = Q^{-1}v$. Thus, A and all of its similar matrices, $Q^{-1}AQ$, have the same eigenvalues. This completes the proof! □

We can now state our theorem regarding the applicability of state feedback!

Theorem 5 ***Stabilization by State Feedback***
If the single input system $\dot{x} = Ax + Bu$ is controllable, then by state feedback $u = -Kx$, the eigenvalues of the closed loop matrix $A - BK$:

$$\dot{x} = Ax - BKx = (A - BK)x \tag{2.27}$$

May be placed arbitrarily by selecting K, which enables us to stabilize the system.

This theorem tells us that as long as $\dot{x} = Ax + Bu$ is a controllable system, when we plug in the input $u = -Kx$, which gives the closed loop system:

$$\dot{x} = Ax + Bu = (A - BK)x \tag{2.28}$$

We can *entirely* control the eigenvalues of $A - BK$ by changing the values of K! This means that we can move all of the eigenvalues of the matrix $A - BK$ to have negative real components, which will guarantee the equilibrium point $x_e = 0$ is globally exponentially stable.
Note that $A - BK$ is called the closed loop matrix, as it gives us the dynamics of the system after we plug our closed loop feedback controller expression in for the input, u.

Proof: We have a controllable linear, time invariant system with a single input:

$$\dot{x} = Ax + Bu,\ x \in \mathbb{R}^n, u \in \mathbb{R} \tag{2.29}$$

Using the lemma from above, since this system is controllable, we may transform it into controllable canonical form to make our analysis much simpler. Defining a coordinate transformation into controllable canonical form $x = Qx_c$, $A_c = Q^{-1}AQ$, $B_c = Q^{-1}Bu$, we get the system:

$$\dot{x}_c = A_c x_c + B_c u \tag{2.30}$$

Now, we define a state feedback control law in these x_c coordinates:

$$u = -K_c x_c = -\begin{bmatrix} k_{c1} & k_{c2} & \dots & k_{cn} \end{bmatrix} x_c \tag{2.31}$$

Substituting this input into our system dynamics, we get:

$$\dot{x}_c = A_c x_c - B_c K_c x_c \tag{2.32}$$
$$\dot{x}_c = (A_c - B_c K_c) x_c \tag{2.33}$$

Now, referring to the forms of A_c and B_c in controllable canonical form, we can show that $A_c - B_c K_c$ is calculated:

$$A_c - B_c K_c = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 - k_1 & -a_1 - k_2 & -a_2 - k_3 & \dots & -a_{n-1} - k_n \end{bmatrix} \tag{2.34}$$

If we can show that by adjusting the values of k_i, we can entirely control the eigenvalues of $A_c - B_c K_c$, we'll have shown that we can make the system $\dot{x}_c = (A_c - B_c K_c)x_c$ stable! Calculating the characteristic polynomial of $A_c - B_c K_c$ in terms of k_i, we get:

$$\lambda^n + (a_{n-1} + k_n)\lambda^{n-1} + ... + (a_1 + k_2)\lambda + (a_0 + k_1) = 0 \tag{2.35}$$

Thus, *every* coefficient of the characteristic polynomial can be changed arbitrarily by modifying the value of k_i! Thus, we can completely decide what the eigenvalues of the closed loop system $\dot{x}_c = (A_c - B_c K_c)x_c$ will be using state feedback. This enables us to choose the eigenvalues to have negative real components, which will stabilize the system.

Now, let's transform this result back to our original coordinates! Recall that we defined $A_c = Q^{-1}AQ$, $B_c = Q^{-1}Bu$, and $x = Qx_c$. Starting with our closed loop system dynamics in controllable canonical form:

$$\dot{x}_c = (A_c - B_c K_c)x_c \tag{2.36}$$

We multiply both sides by Q and substitute $x_c = Q^{-1}x$ to get:

$$\dot{x} = Q(A_c - B_c K_c)x_c \tag{2.37}$$

$$\dot{x} = Q(A_c - B_c K_c)Q^{-1}x \tag{2.38}$$

This expression is now in our original x coordinates. Recall from our lemma above that for any invertible coordinate transformation Q and matrix C, the eigenvalues of $A_c - B_c K_c$ and $Q(A_c - B_c K_c)Q^{-1}$ are the same, as they are similar matrices.

Since we can use K to control the eigenvalues of the system arbitrarily in controllable canonical form, we can also use K to change the eigenvalues of the system arbitrarily in our original coordinates!

Let's try to reduce this expression above to a simpler form to ensure that state feedback in controllable canonical form still corresponds to state feedback in our original coordinates. Multiplying out by Q and Q^{-1}:

$$\dot{x} = QA_cQ^{-1} - QB_cK_cQ^{-1}x \tag{2.39}$$

$$\dot{x} = A - QQ^{-1}BK_cQ^{-1}x \tag{2.40}$$

$$\dot{x} = A - BK_cQ^{-1}x \tag{2.41}$$

If we define $K = K_c Q^{-1}$, we see that our state feedback $u = -K_c x_c$ is equivalent to $u = -Kx$ in our original coordinates, which is the formula for state feedback!

$$\dot{x} = Ax - BKx = (A - BK)x \tag{2.42}$$

Since $A - BK = Q(A_c - B_c K_c)Q^{-1}$, whose eigenvalues we can arbitrarily control, we conclude that using the state feedback $u = -Kx$, we can stabilize the system about its equilibrium point by choosing $(A - BK)$ to have eigenvalues with negative real components.
Thus, state feedback can make $x_e = 0$ a globally exponentially stable equilibrium point as long as the system is controllable. This completes the proof! □

Note that although we analyzed the single input case in the theorem above, the results we developed extend to the multi-input case! In general, as long as $\dot{x} = Ax + Bu$ is a controllable system, we may stabilize it with state feedback.

2.3 Feedback Linearization

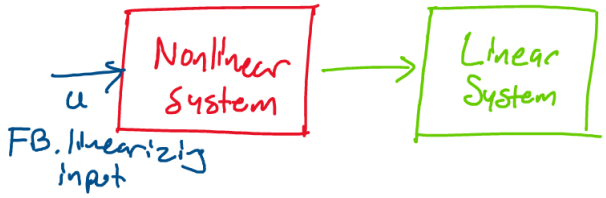
So far, we've talked about a way of controlling the stability of a linear system using a technique known as *state feedback*. We proved that as long as a system is controllable (can be driven between any two states x_0 and x_1 in finite time), we can make the equilibrium point $x_e = 0$ globally exponentially stable using state feedback:

$$u = -Kx \tag{2.43}$$

The simplicity and global exponential guarantees of state feedback make it a highly appealing choice of controller for linear systems!

In robotics, however, most systems that we seek to control are *highly nonlinear*. For instance quadrotors, autonomous vehicles, and robot arms are all described by coupled systems of nonlinear differential equations! This means that we *can't* use plain state feedback, $u = -Kx$ to control these systems in an effective manner!

What can we do to get around this and design controllers with strong performance guarantees for nonlinear systems? For many nonlinear systems, we may perform a process known as **feedback linearization**, where we use feedback control to make a nonlinear system *behave* like a linear system!



Above: Can we find an input u that makes a nonlinear system act linear?

This is an extremely powerful concept in nonlinear control that we'll now work on constructing.

Note that our development in this section will mainly focus on the procedure of developing a feedback linearizing controller, rather than the theoretical conditions for feedback linearizability. For an in-depth analysis of the conditions required for a system to be feedback linearizable, you're encouraged to read chapter 9 of *Nonlinear Systems: Analysis, Stability, and Control* by Sastry.

2.3.1 SISO Feedback Linearization

Let's begin our development of feedback linearizing controllers! In this section, we'll restrict our analysis to the set of control affine single input, single output systems. Note that single input, single output systems are commonly referred to as **SISO** systems.

These control affine SISO systems will have the form:

$$\dot{x} = f(x) + g(x)u,\ x \in \mathbb{R}^n, u \in \mathbb{R} \tag{2.44}$$

$$y = h(x),\ y \in \mathbb{R} \tag{2.45}$$

Recall from our earlier discussion of dynamical systems that the state equation, $\dot{x} = f(x)+g(x)u$, governs the dynamics of the system, while the output equation, $y = h(x)$, describes a term of interest that we wish to control. Note that in our development, we'll always assume that $g(x)$ is nonzero, so there will always be an input in the state equation.

Let's outline the problem of feedback linearization for this system. We want to *somehow* turn this nonlinear system into a linear system just by using feedback control! How can we accomplish this?

A good first step in answering this question might be to focus purely on the output, y. Since y is the quantity we actually wish to control in the system, perhaps it'll be easier to gain a linear relationship by just focusing on the output instead of the entire state vector.

As we'll soon find out, by focusing on the evolution of the system output, we can actually use feedback control to get a linear relationship between input and output!

Let's try and find an expression that encapsulates how the output, y, changes along the trajectories of the system. We can do this by turning the output equation, y, into a differential equation that makes use of the system dynamics, $\dot{x} = f(x) + g(x)u$. Let's try taking the first time derivative of y, and see where that takes us!

Using the chain rule to calculate the first time derivative of $y = h(x)$:

$$\dot{y} = \dot{h}(x) = \frac{\partial h(x)}{\partial x}\dot{x} \tag{2.46}$$

Next, let's substitute in $\dot{x} = f(x) + g(x)u$ for $\dot{x}$ to find out how y changes along the trajectories of the system. This gives us:

$$\dot{y} = \frac{\partial h(x)}{\partial x}(f(x) + g(x)u) \tag{2.47}$$

$$\dot{y} = \frac{\partial h(x)}{\partial x}f(x) + \frac{\partial h(x)}{\partial x}g(x)u \tag{2.48}$$

Interestingly, when we take the derivative of the output, since our original dynamics were affine in u, our output dynamics are affine in u as well! This means that the effect of u on the output can be *easily separated* from the rest of the output dynamics - u is only related to $\dot{y}$ through simple multiplication and addition.

Let's use this simple separation to our advantage, and see if we can find some way to *cancel out* terms in the output dynamics through a clever choice of u. Assuming that $\frac{\partial h}{\partial x}g(x)$ is nonzero, let's pick the following input, and see what effect it has on the system:

$$u = \frac{1}{\frac{\partial h(x)}{\partial x}g(x)}\Big(-\frac{\partial h(x)}{\partial x}f(x) + v\Big) \tag{2.49}$$

Where $v \in \mathbb{R}$ is an arbitrary constant we have total control over. Let's plug this choice of input into the output differential equation and see what results!

$$\dot{y} = \frac{\partial h(x)}{\partial x} f(x) + \frac{\partial h(x)}{\partial x} g(x) u \tag{2.50}$$

$$\dot{y} = \frac{\partial h(x)}{\partial x} f(x) + \frac{\partial h(x)}{\partial x} g(x) \Big[\frac{1}{\frac{\partial h(x)}{\partial x} g(x)} \Big(- \frac{\partial h(x)}{\partial x} f(x) + v \Big) \Big] \tag{2.51}$$

$$\dot{y} = \frac{\partial h(x)}{\partial x} f(x) - \frac{\partial h(x)}{\partial x} f(x) + v \tag{2.52}$$

$$\dot{y} = v \tag{2.53}$$

Miraculously, when we use this input, we get a system with a directly linear relationship between the derivative of the output, $\dot{y}$, and an arbitrary variable that we can control, v!

This input has therefore *transformed* our nonlinear system into a system with a linear relationship between an input, v, and the output, y! Thus, we say that u has **input-output linearized** the system.

Now that we have a linear system, $\dot{y} = v$, we can apply any methods of linear control, for example state feedback, to stabilize or drive the output of the system to a desired value, y_d.

Now that we've found a feedback linearizing input in a simple case, let's take a moment to reflect on what we did and see if there are any holes in our reasoning or improvements we can make. First, let's try to think a little more carefully about the terms:

$$\frac{\partial h(x)}{\partial x} f(x), \ \frac{\partial h(x)}{\partial x} g(x) \tag{2.54}$$

That we use in our input expression. Can we assign a precise mathematical meaning for these terms, and come up with some more convenient notation? After we address these concerns, we can turn our attention to the case where:

$$\frac{\partial h(x)}{\partial x} g(x) = 0 \tag{2.55}$$

For our current choice of feedback linearizing input, where we use the inverse of $\frac{\partial h(x)}{\partial x} g(x)$, we have no way of dealing with the case where this term is zero. Let's discuss some concepts that allow us to deal with these two tasks!

Lie Derivatives

Let's first focus our attention on making the process of feedback linearization a little more mathematically precise. We'll introduce some convenient notation and provide some interpretation of the operations we carry out. When we take the time derivative of the output, we get:

$$\dot{y} = \frac{\partial h(x)}{\partial x} f(x) + \frac{\partial h(x)}{\partial x} g(x) u \tag{2.56}$$

Let's discuss the different terms involved in this expression. Let's start with $\frac{\partial h}{\partial x}$, the spatial gradient of $h(x) : \mathbb{R}^n \to \mathbb{R}$. We define the gradient of $h(x)$ with respect to $x \in \mathbb{R}^n$ as:

$$\frac{\partial h}{\partial x} = \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} & \dots & \frac{\partial h}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \tag{2.57}$$

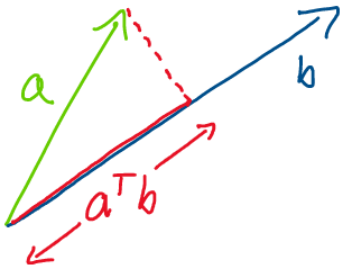
In our expression for the time derivative of y along the trajectories of the system, $\frac{\partial h}{\partial x}$ is multiplied by $f(x)$ and $g(x)$.
What do we know about $f(x)$ and $g(x)$? We know that both $f(x)$ and $g(x)$ take in a vector $x \in \mathbb{R}^n$ and return another vector in $\mathbb{R}^n$! This makes $f(x)$ and $g(x)$ *vector fields* whose value depends on x.
By thinking of $\frac{\partial h}{\partial x}$ as a row vector and $f(x)$ and $g(x)$ as vector fields, we can gain an intuition for the meaning of the products:

$$\frac{\partial h(x)}{\partial x} f(x), \ \frac{\partial h(x)}{\partial x} g(x) \tag{2.58}$$

Recall that if we have two vectors, $a, b \in \mathbb{R}^n$, and we compute:

$$a^T b = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = a_1 b_1 + b_2 b_2 + ... + a_n b_n \tag{2.59}$$

Generally speaking, $a^T b$ helps us describe if and how much a points along the direction of b. For instance, if a and b are orthogonal to one another, $a^T b$ will be zero. On the other hand, in the special case where b is a unit vector, $a^T b$ gives us the length and direction of the projection of a onto b.



Above: If b is a unit vector, $a^T b$ is the length of the projection of a onto b.

Let's apply this concept to the gradient and vector field products we discussed above, and revisit $\frac{\partial h}{\partial x} f(x)$ through the lens of projection:

$$\frac{\partial h(x)}{\partial x} f(x) = \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} & \dots & \frac{\partial h}{\partial x_n} \end{bmatrix} \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix} \tag{2.60}$$

From the expression above, we see that computing $\frac{\partial h}{\partial x} f(x)$ is just like taking the inner product of the gradient of h with respect to x and the vector field $f(x)$! Thus, the product $\frac{\partial h}{\partial x} f(x)$ term tells us about the rate of change of $y = h(x)$ *along* the vector field $f(x)$! Similarly, the product:

$$\frac{\partial h(x)}{\partial x} g(x) \tag{2.61}$$

Tells us about the rate of change of $h(x)$ along the vector field $g(x)$.
In mathematics, these products are referred to as **Lie derivatives**.[2]

Definition 24 ***Lie Derivative***
Given a function $h(x)$ and a vector field $f(x)$, the Lie derivative of $h(x)$ along the vector field $f(x)$ is defined:

$$L_f h(x) = \frac{\partial h(x)}{\partial x} f(x) \tag{2.62}$$

$L_f h(x)$ expresses the rate of change of $h(x)$ along the vector field $f(x)$, as it is the inner product between the gradient of $h(x)$ and the vector field $f(x)$.

Using Lie derivative notation, we may rewrite our expression for the first derivative of y:

$$\dot{y} = \frac{\partial h}{\partial x} f(x) + \frac{\partial h}{\partial x} g(x) u \tag{2.63}$$

$$\dot{y} = L_f h(x) + L_g h(x) u \tag{2.64}$$

Note how Lie derivative notation is much more compact than the expression of partial derivatives! For convenience, we also introduce the following notation for computing higher order Lie derivatives:

$$L_f^2 h(x) = L_f[L_f h(x)] = \frac{\partial (L_f h(x))}{\partial x} f(x) \tag{2.65}$$

$$L_g L_f h(x) = L_g[L_g h(x)] = \frac{\partial (L_f h(x))}{\partial x} g(x) \tag{2.66}$$

Whenever you see a higher order Lie derivative, remember that it's simply a composition of Lie derivatives - we're taking the Lie derivative of a Lie derivative. Using this higher order Lie derivative notation, we can recursively define the p^{th} Lie derivative of $h(x)$ along $f(x)$ as:

$$L_f^p h(x) = L_f[L_f^{p-1} h(x)] = \frac{\partial (L_f^{p-1} h(x))}{\partial x} f(x) \tag{2.67}$$

[2] Pronounced "Lee derivatives."

Let's rewrite our feedback linearizing input in terms of these Lie derivatives. For the single input, single output system $\dot{x} = f(x)+g(x)u,\ y = h(x)$, assuming $L_g h(x) \neq 0$, we define the feedback linearizing input:

$$u = \frac{1}{L_g h(x)}(-L_f h(x) + v) \tag{2.68}$$

Where $v \in \mathbb{R}$ is an arbitrary scalar. Using this notation, it's much easier to see how all of the necessary terms will cancel out when we substitute our input into the output dynamics.

$$\dot{y} = L_f h(x) + L_g h(x) u \tag{2.69}$$

$$\dot{y} = L_f h(x) + L_g h(x) \frac{1}{L_g h(x)}(-L_f h(x) + v) \tag{2.70}$$

$$\dot{y} = L_f h(x) - L_f h(x) + v \tag{2.71}$$

$$\dot{y} = v \tag{2.72}$$

Lie derivative notation will be very convenient for us moving forward, especially as we begin to take higher order derivatives of the output.

High Relative Degree Systems

So far, we've avoided dealing with one major problem in our formulation of a feedback linearizing controller:

$$u = \frac{1}{L_g h(x)}(-L_f h(x) + v) \tag{2.73}$$

What happens if $L_g h(x) = 0$ for all values of x? Firstly, our formula for the input will be undefined! More importantly, based on our expression for the time derivative of y:

$$\dot{y} = L_f h(x) + L_g h(x) u \tag{2.74}$$

If $L_g h(x) = 0$ for all values of x, the input term will completely drop out, and we'll be left with the output dynamics:

$$\dot{y} = L_f h(x) \tag{2.75}$$

Which has no dependence on the input! Since this expression has no input, we have no hope of controlling the value of y through this differential equation. What can we do to reintroduce u into the output dynamics?
Let's try experimenting with the following numerical example. Suppose we have the single input, single output control affine system:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_1 + x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{2.76}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 \tag{2.77}$$

Let's take the first time derivative of y and see if we can find an expression that involves the input! If we can, we'll be able to find a feedback linearizing control input. Note that in this case, since the dynamics and output expressions are simple, it'll be easier just to take the derivative of y directly instead of using Lie derivative notation.

$$\dot{y} = L_f h(x) + L_g h(x) u \tag{2.78}$$
$$\dot{y} = \dot{x}_1 = x_2 \tag{2.79}$$

Since we end up with *just* x_2 and no input, this means that $L_g h(x)$ is zero for all values of x! Let's examine the system dynamics, and see where u appears. Looking at the dynamics, we notice that the expression for $\dot{x}_2$ involves the input! Since $\dot{y} = x_2$, let's try taking another derivative of y!

$$\ddot{y} = \dot{x}_2 = x_1 + x_2 + u \tag{2.80}$$

Now, the input term is nonzero for *all* values of x! We can now define a feedback linearizing control law:

$$u = -(x_1 + x_2) + v \tag{2.81}$$

Such that when we plug u into the formula we calculated for $\ddot{y}$, we get:

$$\ddot{y} = x_1 + x_2 + u \tag{2.82}$$
$$\ddot{y} = x_1 + x_2 - (x_1 + x_2) + v \tag{2.83}$$
$$\ddot{y} = v \tag{2.84}$$

This gives us a linear input-output relationship between v and y! What's more, using the definition of linear controllability, we can prove that this system is controllable, and can therefore use v to drive y to an arbitrary value.
Let's summarize what we learned from this example. We found that even though the input, u, might not appear when we take the first derivative of y, it might appear when we take a higher derivative! Once it appears in a higher derivative term, we can define a feedback linearizing control law that allows us to gain complete control over y.
In general, for arbitrary single input, single output differential equations in control affine form, if $x \in \mathbb{R}^n$, we might have to take up to n derivatives of y:

$$y^{(n)} = \frac{d^n y}{dt^n} \tag{2.85}$$

To get the input term to appear. Keeping this fact in mind, let's generalize our discussion to arbitrary SISO control affine systems. Consider the system:

$$\dot{x} = f(x) + g(x)u,\ x \in \mathbb{R}^n, u \in \mathbb{R} \tag{2.86}$$
$$y = h(x), y \in \mathbb{R} \tag{2.87}$$

Let's try to find an expression for the derivative of y when the input, u, appears! We know that the first time derivative of y is be computed:

$$\dot{y} = \frac{\partial h}{\partial x} f(x) + \frac{\partial h}{\partial x} g(x)u = L_f h(x) + L_g h(x)u \tag{2.88}$$

Suppose that $L_g h(x) = 0$ for all x. This leaves us with:

$$\dot{y} = L_f h(x) \tag{2.89}$$

Following our example from earlier, let's try taking the next derivative of y, $\ddot{y}$, and see if we can get the input to appear! Applying the chain rule, we begin by expressing the second derivative using partial derivatives:

$$\ddot{y} = \frac{\partial}{\partial x}(L_f h(x))\dot{x} \tag{2.90}$$

$$\ddot{y} = \frac{\partial}{\partial x}(L_f h(x))(f(x) + g(x)u) \tag{2.91}$$

$$\ddot{y} = \frac{\partial}{\partial x}(L_f h(x))f(x) + \frac{\partial}{\partial x}(L_f h(x))g(x)u) \tag{2.92}$$

What do we notice about the two terms in this expression? Recall that we defined $L_f h(x), L_g h(x)$ as:

$$L_f h(x) = \frac{\partial h}{\partial x} f(x),\ L_g h(x) = \frac{\partial h}{\partial x} g(x) \tag{2.93}$$

Looking at the expression for $\ddot{y}$, we notice a further Lie derivative in each term! Thus, recalling our notation for taking higher order Lie derivatives, we rewrite $\ddot{y}$ as:

$$\ddot{y} = L_f[L_f h(x)] + L_g[L_f h(x)]u \tag{2.94}$$

$$\ddot{y} = L_f^2 h(x) + L_g L_f h(x)u \tag{2.95}$$

What happens if $L_g L_f h(x)$ is *also* zero for all x? We can continue taking higher and higher derivatives of y until finally, the input shows up!
Suppose that the input term is zero for all derivatives of y up to the r^{th} derivative, where $r \leq n$. Since the input term is zero at the $r-1$ derivative, $y^{(r-1)}$ is computed:

$$y^{(r-1)} = L_f^{r-1} h(x) \tag{2.96}$$

Since there is no input term, we take the derivative of y once again:

$$y^{(r)} = \frac{\partial}{\partial x}(L_f^{r-1} h(x))\dot{x} \tag{2.97}$$

$$y^{(r)} = \frac{\partial}{\partial x}(L_f^{r-1} h(x))(f(x) + g(x)u) \tag{2.98}$$

$$y^{(r)} = \frac{\partial}{\partial x}(L_f^{r-1} h(x))f(x) + \frac{\partial}{\partial x}(L_f^{r-1} h(x))g(x)u \tag{2.99}$$

$$y^{(r)} = L_f^r h(x) + L_g L_f^{r-1} h(x)u \tag{2.100}$$

Imagine that *finally*, after taking the r^{th} derivative of y, the input term:

$$L_g L_f^{r-1} h(x) \neq 0 \tag{2.101}$$

Is nonzero. For the r^{th} derivative, we are therefore able to calculate a feedback linearizing control law. We pick an input:

$$u = \frac{1}{L_g L_f^{r-1} h(x)} (-L_f^r h(x) + v) \tag{2.102}$$

Where $v \in \mathbb{R}$ is an arbitrary scalar that we can control. When substituting this control law into the expression for $y^{(r)}$, we get:

$$y^{(r)} = L_f^r h(x) + L_g L_f^{r-1} h(x) \frac{1}{L_g L_f^{r-1} h(x)} (-L_f^r h(x) + v) \tag{2.103}$$

$$y^{(r)} = L_f^r h(x) - L_f^r h(x) + v \tag{2.104}$$

$$y^{(r)} = v \tag{2.105}$$

Thus, after having taken r derivatives of the output, we have successfully found a feedback linearizing control law that gives us a linear relationship between the output, y, and an input, v.
The smallest number of derivatives, r, that we need to take of the output before the input appears is called the **relative degree** of the system. We can define relative degree more formally using Lie derivative notation.

Definition 25 ***Relative Degree***
The SISO system $\dot{x} = f(x) + g(x)u, y = h(x), x \in \mathbb{R}^n$ has a relative degree r if:

$$L_g L_f^{p-1} h(x) = 0, \; p < r \tag{2.106}$$

$$L_g L_f^{r-1} h(x) \neq 0 \tag{2.107}$$

This makes r the smallest number of time derivatives of the output y that must be taken for the input to appear. Note that if $x \in \mathbb{R}^n$ and $g(x) \neq 0$, then $r \leq n$.

Let's summarize what we learned in this section with a general procedure for dealing with high relative degree systems. In this procedure, we'll refer to a SISO control affine system $\dot{x} = f(x) + g(x)u, \; x \in \mathbb{R}^n$, and will assume that $g(x) \neq 0$.
To find a feedback linearizing control law u, we can:

1. Take the first time derivative of the output, y:

$$\dot{y} = L_f h(x) + L_g h(x) u \tag{2.108}$$

If $L_g h(x) \neq 0$, skip to step 3. If $L_g h(x) = 0$, continue to step 2.

2. Continue taking derivatives of the output, y, until you reach the relative degree of the system, r, where the input appears in the expression for the derivative.

$$y^{(r)} = L_f^r h(x) + L_g L_f^{r-1} h(x) u \tag{2.109}$$

   At the r^{th} derivative, $L_g L_f^{r-1} h(x) \neq 0$.

3. Define a feedback linearizing control law:

$$u = \frac{1}{L_g L_f^{r-1} h(x)} (-L_f^r h(x) + v) \tag{2.110}$$

   Where $v \in \mathbb{R}$ is an arbitrary scalar we may use to control the output.

4. Apply the methods of linear control design to control the resulting system:

$$y^{(r)} = v \tag{2.111}$$

Tracking Control

Let's discuss some control design techniques for a SISO input-output linearized system:

$$y^{(r)} = v \tag{2.112}$$

How can we choose $v \in \mathbb{R}$ such that y follows a desired trajectory, $y_d(t)$? Let's see if we can devise an input by driving the error between y and y_d to zero! We define an error vector, ε:

$$\varepsilon = \begin{bmatrix} e \\ \dot{e} \\ \vdots \\ e^{(r-1)} \end{bmatrix} = \begin{bmatrix} y - y_d \\ \dot{y} - \dot{y}_d \\ \vdots \\ y^{(r-1)} - y_d^{(r-1)} \end{bmatrix} \tag{2.113}$$

Using this error vector, let's experiment with the following choice of v:

$$v = -K\varepsilon + y_d^{(r)} \tag{2.114}$$

Where $K = [k_1, k_2, ..., k_r]$ is a gain matrix and $y_d^{(r)}$ is the r^{th} derivative of the desired trajectory. Let's plug this choice of v into the feedback linearized system and see what we get!

$$y^{(r)} = -K\varepsilon + y_d^{(r)} \tag{2.115}$$

$$y^{(r)} - y_d^{(r)} = -K\varepsilon \tag{2.116}$$

$$e^{(r)} = -K\varepsilon \tag{2.117}$$

We now have an r^{th} order differential equation in tracking error, e, and its derivatives. Can we choose k_i to drive the tracking error to zero? Let's try and reduce this problem to a state feedback problem to find out!
First, we'll rewrite this high order differential equation as a system of r first order differential equations. Using our definition of e from above:

$$\begin{bmatrix} \dot{e} \\ \ddot{e} \\ \dddot{e} \\ \vdots \\ \dot{e}^{(r)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -k_1 & -k_2 & -k_3 & \dots & -k_r \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \\ \ddot{e} \\ \vdots \\ e^{(r-1)} \end{bmatrix} \tag{2.118}$$

We know that if we can choose k_i such that the eigenvalues of the matrix in the equation above have all of their real components less than zero, we can asymptotically drive the error of the system to zero!
Computing the characteristic polynomial of the system, we have:

$$\lambda^r + k_r\lambda^{r-1} + ... + k_2\lambda + k_1 = 0 \tag{2.119}$$

Thus, we can arbitrarily change the coefficients of the characteristic polynomial using k_i, and can move the coefficients such that the eigenvalues of the matrix all have negative real components. This means that using this choice of v, we can make the tracking error of the system decay asymptotically to zero!
Take a moment to notice the similarities between our approach in this section and our approach when designing state feedback controllers to stabilize systems. Here, instead of finding an input u to stabilize a state vector to zero, we find an input v to stabilize an error vector to zero.
It's extremely important to note that the process of feedback linearization isn't perfect! Not all systems will be feedback linearizable in this manner, and certain systems, known as non-minimum phase systems, can sometimes give these input-output linearized systems some undesirable qualities.

2.3.2 MIMO Feedback Linearization

So far, we've constrained our discussion of feedback linearization to single input, single output systems. Let's generalize the concepts we've developed thus far to multi input, multi output **(MIMO)** systems. In this case, both the input, u, and the output, y, will be vectors instead of simple scalar numbers!
Within the set of MIMO systems, we'll focus on the class of **square** control affine systems, which are control affine systems with the same number of inputs as outputs. These systems will be of the form:

$$\dot{x} = f(x) + g(x)u,\ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{2.120}$$
$$y = h(x),\ y \in \mathbb{R}^m \tag{2.121}$$

Thankfully, the procedure for finding feedback linearizing controllers for multi input, multi output systems is quite similar to the SISO case!

We can reason about this similarity by thinking about the output vector, $y \in \mathbb{R}^m$, as a collection of scalar outputs, $y_i(t)$.

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_m(x) \end{bmatrix} \in \mathbb{R}^m \tag{2.122}$$

In each entry, we have a relationship of the form $y_i = h_i(x)$, which is of a similar form to the single output case we discussed in the previous section.
Let's break up this problem one element at time, and focus on an arbitrary entry of the output vector, y_j. As with the SISO case, we'll start by taking the derivatives of y_j along the trajectories of the system.
Taking the first time derivative of y_j and expanding:

$$\dot{y}_j = \frac{\partial h_j(x)}{\partial x}\dot{x} = \frac{\partial h_j(x)}{\partial x}f(x) + \frac{\partial h_j}{\partial x}g(x)u \tag{2.123}$$

Let's think about the different components of this derivative! The first component:

$$\frac{\partial h_j}{\partial x}f(x) = L_f h_j(x) \in \mathbb{R} \tag{2.124}$$

Is the familiar Lie derivative from the single input, single output case! Thus, we may use our established Lie derivative notation in its calculation. What about the second component?

$$\frac{\partial h_j}{\partial x}g(x) \tag{2.125}$$

In the multi input, multi output case, the second component works a little differently to the SISO case. Now, instead of being a vector, $g(x) \in \mathbb{R}^{n\times m}$ is a matrix-valued valued function!

$$g(x) = \begin{bmatrix} | & | & & | \\ g_1(x) & g_2(x) & \dots & g_m(x) \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n\times m} \tag{2.126}$$

Let's see what effect this matrix has on the computation of the second term. Multiplying each column of $g(x)$ by the gradient, we see that we can express the product of $\frac{\partial h_j}{\partial x}$ and $g(x)$ using Lie derivatives.

$$\frac{\partial h_j}{\partial x}g(x) = \frac{\partial h_j}{\partial x}\begin{bmatrix} | & | & & | \\ g_1(x) & g_2(x) & \dots & g_m(x) \\ | & | & & | \end{bmatrix} \tag{2.127}$$

$$\frac{\partial h_j}{\partial x}g(x) = \begin{bmatrix} \frac{\partial h_j}{\partial x}g_1(x) & \frac{\partial h_j}{\partial x}g_2(x) & \dots & \frac{\partial h_j}{\partial x}g_m(x) \end{bmatrix} \tag{2.128}$$

$$\frac{\partial h_j}{\partial x}g(x) = \begin{bmatrix} L_{g_1}h_j(x) & L_{g_2}h_j(x) & ... & L_{g_m}h_j(x) \end{bmatrix} \tag{2.129}$$

Combining the expressions for the first and second components of the derivative, we find that $\dot{y}_j$ may be expressed in Lie derivative notation as:

$$\dot{y}_j = \frac{\partial h_j(x)}{\partial x}\dot{x} = \frac{\partial h_j(x)}{\partial x}f(x) + \frac{\partial h_j}{\partial x}g(x)u \tag{2.130}$$

$$\dot{y}_j = L_f h_j(x) + \begin{bmatrix} L_{g_1}h_j(x) & L_{g_2}h_j(x) & ... & L_{g_m}h_j(x) \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \tag{2.131}$$

$$\dot{y}_j = L_f h_j(x) + L_{g_1}h_j(x)u_1 + L_{g_2}h_j(x)u_2 + ... + L_{g_m}h_j(x)u_m \tag{2.132}$$

We can write this final expression more compactly using summation notation:

$$\dot{y}_j = L_f h_j(x) + \sum_{i=1}^{m} L_{g_i}h_j(x)u_i \tag{2.133}$$

From this expression, we observe that the time derivative of each individual output has a similar form to the single input, single output case! Instead of having just one input term, however, we have to sum up m terms to account for the effect each of the m inputs has on $\dot{y}_j$.

Just like with SISO systems, MIMO systems have the possibility that an input might not show up in the expression for the first derivative of the output! That's to say, for each $\dot{y}_j$, the term:

$$\sum_{i=1}^{m} L_{g_i}h_j(x)u_i \tag{2.134}$$

Might be zero when we take the first derivative of y_j! As with before, we can continue taking higher and higher derivatives of each $\dot{y}_j$ until at least one input, u_i, appears. Let's name the smallest derivative of y_j for which at least one input appears r_j.

In similar fashion to the SISO case, we may show that the r_j derivative of y_j is computed:

$$y_j^{(r_j)} = L_f^{r_j}h_j(x) + \sum_{i=1}^{m} L_{g_i}L_f^{r_j-1}h_j(x)u_i \tag{2.135}$$

Where at least one of the terms in the sum $\sum_{i=1}^{m} L_{g_i}L_f^{r_j-1}h_j(x)u_i$ is nonzero.

Now that we've examined the behavior of an individual y_j, let's bring together all of the entries of the output vector to form a system of differential equations that describe the evolution of y.

As we do this, always keep in mind - what we're doing here is almost exactly the same as the SISO case! We're simply performing the same calculations for each element of the output vector.

Using our notation from above that r_j is the first derivative of y_j for which at

least one input appears in $y_j^{(r_j)}$, we may place each derivative $y_j^{(r_j)}$ in a vector as follows:

$$\begin{bmatrix} y_1^{(r_1)} \\ y_2^{(r_2)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = \begin{bmatrix} L_f^{r_1} h_1(x) + \sum_{i=1}^m L_{g_i} L_f^{r_1-1} h_1(x) u_i \\ L_f^{r_2} h_2(x) + \sum_{i=1}^m L_{g_i} L_f^{r_2-1} h_2(x) u_i \\ \vdots \\ L_f^{r_m} h_m(x) + \sum_{i=1}^m L_{g_i} L_f^{r_m-1} h_m(x) u_i \end{bmatrix} \tag{2.136}$$

Recall that since we are using the r_j derivative of each y_j, there is at least one input in each row of the expression above!
Let's try factoring out the input vector to better examine the different components of this relation. We can begin by splitting the expression into two vectors, one which contains input terms and the other which does not.

$$\begin{bmatrix} y_1^{(r_1)} \\ y_2^{(r_2)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = \begin{bmatrix} L_f^{r_1} h_1(x) \\ L_f^{r_2} h_2(x) \\ \vdots \\ L_f^{r_m} h_m(x) \end{bmatrix} + \begin{bmatrix} \sum_{i=1}^m L_{g_i} L_f^{r_1-1} h_1(x) u_i \\ \sum_{i=1}^m L_{g_i} L_f^{r_2-1} h_2(x) u_i \\ \vdots \\ \sum_{i=1}^m L_{g_i} L_f^{r_m-1} h_m(x) u_i \end{bmatrix} \tag{2.137}$$

Notice how in each row of the vector containing inputs, the inputs are multiplied by the Lie derivatives of the system in a similar manner! Let's take advantage of this structure to break up the input vector into a matrix-vector product!

$$\begin{bmatrix} y_1^{(r_1)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = \begin{bmatrix} L_f^{r_1} h_1 \\ \vdots \\ L_f^{r_m} h_m \end{bmatrix} + \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1 & \dots & L_{g_m} L_f^{r_1-1} h_1 \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f^{r_m-1} h_m & \dots & L_{g_m} L_f^{r_m-1} h_m \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \tag{2.138}$$

The matrix being multiplied by the input vector, which has the shape $m \times m$, is commonly referred to as the **decoupling matrix**, as it separates the effect of the input from the other output dynamics.
For convenience, let's name the decoupling matrix $A(x)$:

$$A(x) = \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1 & \dots & L_{g_m} L_f^{r_1-1} h_1 \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f^{r_m-1} h_m & \dots & L_{g_m} L_f^{r_m-1} h_m \end{bmatrix} \in \mathbb{R}^{m \times m} \tag{2.139}$$

Using this matrix, we may extend the concept of relative degree from SISO systems to MIMO systems! Recall that for a SISO system, the relative degree of a system is the smallest derivative r such that:

$$y^{(r)} = L_f^r h(x) + L_g L_f^{r-1} h(x) u \tag{2.140}$$

Has nonzero $L_g L_f^{r-1} h(x)$ for some values of x. The fact that this term was nonzero allowed us to take its multiplicative inverse when defining a SISO feedback linearizing control law.

Inspired by the idea of being able to *invert* the input term in the output dynamics, we define **vector relative degree**, a concept that generalizes relative degree to MIMO systems.

Definition 26 ***Vector Relative Degree***
The vector relative degree at $x_0 \in \mathbb{R}^n$ of a square, control affine MIMO system:

$$\dot{x} = f(x) + g(x)u,\ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{2.141}$$
$$y = h(x),\ y \in \mathbb{R}^m \tag{2.142}$$

Is the collection of the smallest values $(r_1, r_2, ..., r_m)$ such that when $x = x_0$:

$$L_{g_i} L_f^{r_i - 1} h_i(x) \neq 0,\ 1 \leq i \leq m \tag{2.143}$$

And the decoupling matrix:

$$A(x) = \begin{bmatrix} L_{g_1} L_f^{r_1 - 1} h_1 & \dots & L_{g_m} L_f^{r_1 - 1} h_1 \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f^{r_m - 1} h_m & \dots & L_{g_m} L_f^{r_m - 1} h_m \end{bmatrix} \tag{2.144}$$

Is invertible at $x = x_0$.

Note that the invertibility of the decoupling matrix *can* depend on the state of the system! This is why we specify that the vector relative degree of a system is defined "at a point."
Now that we're equipped with an understanding of vector relative degree for MIMO systems, we may tackle the task of designing a MIMO feedback linearizing controller. Note that for the remainder of this section, we'll assume that the vector relative degree of the system is well-defined, and that we can actually find r_1 through r_m that make $A(x)$ invertible.
If the system has a vector relative degree of $(r_1, ..., r_m)$, we know that we can express the derivatives of each element of the output as:

$$\begin{bmatrix} y_1^{(r_1)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = \begin{bmatrix} L_f^{r_1} h_1 \\ \vdots \\ L_f^{r_m} h_m \end{bmatrix} + \begin{bmatrix} L_{g_1} L_f^{r_1 - 1} h_1 & \dots & L_{g_m} L_f^{r_1 - 1} h_1 \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f^{r_m - 1} h_m & \dots & L_{g_m} L_f^{r_m - 1} h_m \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \tag{2.145}$$

Where the decoupling matrix $A(x)$ is invertible. For convenience, let's name the first term b. This allows us to express the above as:

$$\begin{bmatrix} y_1^{(r_1)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = b + A(x)u \tag{2.146}$$

Let's find a feedback linearizing input u that gives a linear relationship between an arbitrary input vector v and the derivatives of the output. Assuming that $A(x)$ is invertible in our region of interest, we pick the input:

$$u = A^{-1}(x)(-b + v) \tag{2.147}$$

Where $v \in \mathbb{R}^m$ is an arbitrary vector we can completely control. Plugging this input into the system:

$$\begin{bmatrix} y_1^{(r_1)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = b + A(x)A^{-1}(x)(-b + v) \tag{2.148}$$

$$\begin{bmatrix} y_1^{(r_1)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = v \tag{2.149}$$

Thus, we have successfully arrived at a linear relationship between an input vector, v, and the derivatives of the output vector! This choice of u has therefore input-output linearized the MIMO system!
Notice the similarity of the MIMO input to the SISO input! Just as with the SISO case, we take the product of the inverse of the input dynamics with the sum of the opposite of the L_f terms and an arbitrary vector v.

$$A^{-1}(x)(-b + v)b \iff \frac{1}{L_g L_f^{r-1} h(x)}(-L_f^r h(x) + v) \tag{2.150}$$

$$\textit{MIMO} \iff \textit{SISO} \tag{2.151}$$

Let's summarize our procedure for MIMO feedback linearization with a step by step procedure.

1. Take the time derivative of each term in the output vector, y_j, until at least one input variable appears. Name the derivative at which the input appears r_j. Using Lie derivative notation, this derivative is computed:

$$y^{(r_j)} = L_f^{r_j} h_j(x) + \sum_{i=1}^{m} L_{g_i} L_f^{r_j - 1} h_j(x) u_i \tag{2.152}$$

   Once the Lie derivatives have all been computed in this manner, proceed to step 2.

2. Verify that the matrix:

$$A(x) = \begin{bmatrix} L_{g_1} L_f^{r_1 - 1} h_1 & \dots & L_{g_m} L_f^{r_1 - 1} h_1 \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f^{r_m - 1} h_m & \dots & L_{g_m} L_f^{r_m - 1} h_m \end{bmatrix} \tag{2.153}$$

Is invertible. If the sum of the derivatives $r_1 + r_2 + ... + r_m$ exceeds n, the size of the state vector, and $A(x)$ is *still* not invertible, see the next section on dynamic extension. Otherwise, continue to step 3.

3. Assuming $A(x)$ is invertible in our region of interest, choose a feedback linearizing control input:

$$u = A^{-1}(x)(-b + v) \tag{2.154}$$

Where b is the vector filled with $L_f^{r_i} h_i$ terms and $v \in \mathbb{R}^m$ is an arbitrary vector we may use for control.

4. Apply the methods of linear control design to choose v for the resulting linear system:

$$\begin{bmatrix} y_1^{(r_1)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = v \tag{2.155}$$

2.3.3 Dynamic Extension

So far, we've avoided explicitly dealing with the case when $A(x)$ is not an invertible matrix and cannot be made invertible by taking higher derivatives of the output.
This situation can happen if we find the matrix of Lie derivatives $A(x)$ to have an entire column of zeros, for example:

$$A(x) = \begin{bmatrix} L_{g_1} L_f^{r_1 - 1} h_1 & \dots & L_{g_{m-1}} L_f^{r_1 - 1} h_1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ L_{g_1} L_f^{r_m - 1} h_m & \dots & L_{g_{m-1}} L_f^{r_m - 1} h_m & 0 \end{bmatrix} \tag{2.156}$$

What does this tell us about the system? This means that even as we take higher derivatives, there is *no way* for the the input to show up in the dynamics! How can we resolve this?
If the input doesn't show up when taking derivatives of the output dynamics, it's possible that one set of inputs in the system show up at a much earlier derivative of the output than other inputs!
For instance, consider the following scenario, where we have a two input, two output system. Imagine that when we take the first derivative of y, we get the relationship:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.157}$$

From the process we outlined in the previous section, we have the appearance of at least one input in each derivative of y, so in theory, we *should* be done with differentiating the output. However, $A(x)$ has a column of zeros, and is

therefore not invertible!
It's possible that although the second input, u_2, doesn't have any relationship with the first derivative of y, it *might* have a relationship with a second or higher derivative! We're now faced with the following challenge: u_1 appears in the first derivative of y, but u_2 might appear in a higher derivative!
If u_1 appears in a lower derivative but u_2 in a higher derivative, we could end up with a relationship such as the following, where u_2 and a derivative of u_1 appear in the second derivative of the outputs:

$$\ddot{y}_1 = \dot{u}_1 + 3u_2 \tag{2.158}$$
$$\ddot{y}_2 = \dot{u}_1 + 5u_2 \tag{2.159}$$

We need to somehow devise a method to *slow down* the appearance of u_1 so it appears at the same level of derivative as u_2! If we can slow down the appearance of u_1 in this manner, we'll get both inputs appearing at the same level of derivative, and will therefore be able to invert $A(x)$.
How can we perform this slowing down operation? We see that in the example above, we had an appearance of $\dot{u}_1$ in the expression for the higher derivative. Thus, if we define a *new* input to the system:

$$v_1 = \dot{u}_1 \tag{2.160}$$

And control v_1 instead of u_1, we will get the following output dynamics:

$$\ddot{y}_1 = v_1 + 3u_2 \tag{2.161}$$
$$\ddot{y}_2 = v_1 + 5u_2 \tag{2.162}$$
$$\begin{bmatrix}\ddot{y}_1 \\ \ddot{y}_2\end{bmatrix} = \begin{bmatrix}1 & 3 \\ 1 & 5\end{bmatrix}\begin{bmatrix}v_1 \\ u_2\end{bmatrix} \tag{2.163}$$

By controlling the derivative of u_1, $\dot{u}_1 = v_1$. instead of directly controlling u_1, we can therefore find an invertible linear relationship between an input vector:

$$\begin{bmatrix}v_1 \\ u_2\end{bmatrix} \tag{2.164}$$

And the derivatives of the output vector. We can then use this relationship to design a feedback linearizing controller! This method of controlling a derivative of the input instead of the input itself is called **dynamic extension**. We can use dynamic extension to synchronize the appearance of inputs, and ensure we have an invertible relationship.
Let's formalize all of the steps involved in the process of dynamic extension with a practical example: a planar quadrotor. Using the methods of Lagrangian or Newtonian mechanics, we can show that the dynamics of a planar quadrotor

may be expressed in state space form as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} x_2 \\ 0 \\ x_4 \\ -g \\ x_6 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -\frac{1}{m}\sin x_5 & 0 \\ 0 & 0 \\ \frac{1}{m}\cos x_5 & 0 \\ 0 & 0 \\ 0 & \frac{1}{I_{xx}} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.165}$$

Where x_1, x_3 are the x, z coordinates of the quadrotor, x_5 is the orientation angle of the quadrotor, and x_2, x_4, x_6 are the rates of each of these quantities. Let's define the outputs of the system to be the x and z coordinates of the quadrotor:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} \tag{2.166}$$

Let's follow our established process of MIMO feedback linearization, and begin taking the derivatives of the outputs until at least one input appears in each derivative. Note that in this example, it'll be easier to take the derivatives of the output directly instead of using Lie derivative notation.
Taking the first derivatives of y_1 and y_2:

$$\dot{y}_1 = \dot{x}_1 = x_2 \tag{2.167}$$
$$\dot{y}_2 = \dot{x}_3 = x_4 \tag{2.168}$$

No input terms have showed up yet! Let's take another derivative and see what happens. Using the dynamics of the quadrotor, we find:

$$\ddot{y}_1 = \dot{x}_2 = -\frac{u_1}{m}\sin x_5 \tag{2.169}$$
$$\ddot{y}_2 = \dot{x}_4 = \frac{u_1}{m}\cos x_5 \tag{2.170}$$

Now, we have at least one input term in each output derivative! Following our established MIMO procedure, we factor out the input in matrix form as:

$$\begin{bmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{m}\sin x_5 & 0 \\ \frac{1}{m}\cos x_5 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.171}$$

As we can see, in our expression $\ddot{y} = A(x)u$, $A(x)$ has an entire column of zeros! We observe that the input u_1 has shown up in a derivative *before* u_2 has had the chance to appear! Using the process we discussed above, we now want to *slow down* the appearance of u_1 so we can have all of the inputs appearing at the same derivative of y. Let's define a new input:

$$v_1 = \dot{u}_1 \tag{2.172}$$

Instead of trying to control u_1 directly, we'll try to control its derivative, v_1. However, since u_1 is *still* the actual input to the system, we need some way to

keep track of its actual value as we change its derivative. Only this way will we know what u_1 we can actually send to the system!
To keep track of the value of u_1, we define a new *extended* state vector by adding u_1 to the end of the original state vector:

$$\tilde{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ u_1 \end{bmatrix} \tag{2.173}$$

This will enable us to keep track of the value of u_1 as we control its derivative. By adding u_1 to the state vector, we have *extended* the dynamics of our original system! This is what gives the process of dynamic extension its name.
Let's rewrite our original system dynamics in terms of our new, extended state vector. We can describe $\dot{\tilde{x}}$ by:

$$\dot{\tilde{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{u}_1 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{u_1}{m}\sin x_5 \\ x_4 \\ \frac{u_1}{m}\cos x_5 - g \\ x_6 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{I_{xx}} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ u_2 \end{bmatrix} \tag{2.174}$$

Notice that since u_1 is part of our new state vector, $\tilde{x}$, this extended system *still* follows the form:

$$\dot{\tilde{x}} = f(\tilde{x}) + g(\tilde{x})w \tag{2.175}$$

Where $w = [v_1, u_2]^T$ is our new, modified input vector to the system. We now include u_1 in $f(\tilde{x})$ rather than in $g(\tilde{x})w$ since we treat u_1 like any other element of the state vector $\tilde{x}$. Now that we've slowed the appearance of u_1 and have extended the dynamics of our system, let's try using this modified system to find a feedback linearizing control law!
As always, we start by taking the derivatives of the outputs, $y_1 = x_1, y_2 = x_3$. Note that the outputs don't change when we change the state vector.
What *will* change is that when taking the derivative of the outputs, we'll look for the appearance of at least one element from our *new* input vector, $w = [v_1, u_2]$, rather than an element from our *old* input vector, $u = [u_1, u_2]$. Using the dynamics as reference:

$$\dot{y}_1 = \dot{x}_1 = x_2 \tag{2.176}$$
$$\dot{y}_2 = \dot{x}_3 = x_4 \tag{2.177}$$

As neither v_1 nor u_2 appear, we take another derivative:

$$\ddot{y}_1 = \dot{x}_2 = -\frac{u_1}{m}\sin x_5 \tag{2.178}$$

$$\ddot{y}_2 = \dot{x}_4 = \frac{u_1}{m}\cos x_5 - g \tag{2.179}$$

In this step, it's extremely import to remember - u_1 is now part of our state vector, not our input vector - we don't consider it to be an input! Thus, we must take another derivative. Using the product and chain rules:

$$y_1^{(3)} = \ddot{x}_2 = -\frac{\dot{u}_1}{m}\sin x_5 - \frac{u_1}{m}\dot{x}_5\cos x_5 \tag{2.180}$$

$$y_2^{(3)} = \ddot{x}_4 = \frac{\dot{u}_1}{m}\cos x_5 - \frac{u_1}{m}\dot{x}_5\sin x_5 \tag{2.181}$$

Now, we can substitute $\dot{u}_1 = v$ and $\dot{x}_5 = x_6$ and rewrite this in matrix form as:

$$\begin{bmatrix} y_1^{(3)} \\ y_2^{(3)} \end{bmatrix} = \begin{bmatrix} -\frac{u_1}{m}x_6\cos x_5 \\ -\frac{u_1}{m}x_6\sin x_5 \end{bmatrix} + \begin{bmatrix} -\frac{1}{m}\sin x_5 & 0 \\ \frac{1}{m}\cos x_5 & 0 \end{bmatrix}\begin{bmatrix} v_1 \\ u_2 \end{bmatrix} \tag{2.182}$$

Now, we see that we have the same problem as earlier! We still have a column of zeros and thus haven't yet slowed down the appearance of u_1 enough! We therefore repeat the process of dynamic extension, and aim to control the second derivative of u_1 instead of the first to slow its appearance down further. We define a new input:

$$v_2 = \dot{v}_1 = \ddot{u}_1 \tag{2.183}$$

Now, in addition to keeping track of the value of u_1, we must keep track of the value of $v_1 = \dot{u}_1$, as we will now be controlling the value of $\dot{v}_1 = \ddot{u}_1$. Appending $v_1 = \dot{u}_1$ to $\tilde{x}$, we redefine our extended state vector as:

$$\tilde{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ u_1 \\ v_1 \end{bmatrix} \tag{2.184}$$

We also redefine our augmented input vector, $w = [w_1, w_2] = [v_2, u_2]$. Using these new state and input vectors, we once again rewrite our system in the form:

$$\dot{\tilde{x}} = f(\tilde{x}) + g(\tilde{x})w \tag{2.185}$$

Taking the derivative of the newly modified state vector and looking at our original dynamics, we find:

$$\dot{\tilde{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{u}_1 \\ \dot{v}_1 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{u_1}{m}\sin x_5 \\ x_4 \\ \frac{u_1}{m}\cos x_5 - g \\ x_6 \\ 0 \\ v_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{I_{xx}} \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix} \tag{2.186}$$

Note that because v_1 is now part of our extended state vector, $\tilde{x}$, this system is in the correct form, $\dot{\tilde{x}} = f(\tilde{x}) + g(\tilde{x})w$. Once again, remember that our outputs, $y_1 = x_1, y_2 = x_3$, are *not* affected by extending the state vector.

Let's try performing our steps of feedback linearization one more time with this further extended state vector. This time, we'll find that we'll get an invertible matrix $A(x)$! Taking the derivatives of y_1, y_2 until one of the inputs in w appears:

$$\dot{y}_1 = \dot{x}_1 = x_2 \tag{2.187}$$
$$\dot{y}_2 = \dot{x}_3 = x_4 \tag{2.188}$$
$$\ddot{y}_1 = \dot{x}_2 = -\frac{u_1}{m}\sin x_5 \tag{2.189}$$
$$\ddot{y}_2 = \dot{x}_4 = \frac{u_1}{m}\cos x_5 \tag{2.190}$$
$$y_1^{(3)} = \ddot{x}_2 = -\frac{v_1}{m}\sin x_5 - \frac{u_1}{m}x_6\cos x_5 \tag{2.191}$$
$$y_2^{(3)} = \ddot{x}_4 = \frac{v_1}{m}\cos x_5 - \frac{u_1}{m}x_6\sin x_5 \tag{2.192}$$

Finally, when we take the fourth derivative of y_1 and y_2, we get:

$$y_1^{(4)} = -\frac{\dot{v}_1}{m}\sin x_5 - \frac{v_1}{m}\dot{x}_5\cos x_5 - \frac{\dot{u}_1}{m}x_6\cos x_5 \tag{2.193}$$
$$-\frac{u_1}{m}\dot{x}_6\cos x_5 + \frac{u_1}{m}x_6\dot{x}_5\sin x_5 \tag{2.194}$$
$$y_1^{(4)} = -\frac{v_2}{m}\sin x_5 - \frac{v_1}{m}x_6\cos x_5 - \frac{v_1}{m}x_6\cos x_5 \tag{2.195}$$
$$-\frac{u_1}{m}\frac{u_2}{I_{xx}}\cos x_5 + \frac{u_1}{m}x_6^2\sin x_5 \tag{2.196}$$
$$y_2^{(4)} = \frac{\dot{v}_1}{m}\cos x_5 + \frac{v_1}{m}\dot{x}_5\sin x_5 - \frac{\dot{u}_1}{m}x_6\sin x_5 \tag{2.197}$$
$$-\frac{u_1}{m}\dot{x}_6\sin x_5 - \frac{u_1}{m}x_6\dot{x}_5\cos x_5 \tag{2.198}$$
$$y_2^{(4)} = \frac{v_2}{m}\cos x_5 + \frac{v_1}{m}x_6\sin x_5 - \frac{v_1}{m}x_6\sin x_5 \tag{2.199}$$
$$-\frac{u_1}{m}\frac{u_2}{I_{xx}}\sin x_5 - \frac{u_1}{m}x_6^2\cos x_5 \tag{2.200}$$

Rewriting this in matrix form and factoring out the two inputs, $w_1 = v_2, w_2 = u_2$, we get:

$$\begin{bmatrix} y_1^{(4)} \\ y_2^{(4)} \end{bmatrix} = \begin{bmatrix} -\frac{v_1}{m} x_6 \cos x_5 - \frac{v_1}{m} x_6 \cos x_5 + \frac{u_1}{m} x_6^2 \sin x_5 \\ \frac{v_1}{m} x_6 \sin x_5 - \frac{v_1}{m} x_6 \sin x_5 - \frac{u_1}{m} x_6^2 \cos x_5 \end{bmatrix} \tag{2.201}$$

$$+ \begin{bmatrix} -\frac{1}{m} \sin x_5 & -\frac{u_1}{mI_{xx}} \cos x_5 \\ \frac{v_2}{m} \cos x_5 & -\frac{u_1}{mI_{xx}} \sin x_5 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix} \tag{2.202}$$

Thus, as long as we don't set $u_1 = 0$, this matrix will be invertible for many values of x. After two rounds of dynamic extension, we have therefore *finally* found an $A(x)$ that we can invert! The dynamics above may be rewritten:

$$\begin{bmatrix} y_1^{(4)} \\ y_2^{(4)} \end{bmatrix} = b + A(x)w \tag{2.203}$$

If we choose w to be the following:

$$w = A^{-1}(-b + v) \tag{2.204}$$

Where v is an arbitrary vector in $\mathbb{R}^m$, we can feedback linearize the relationship between the input and the output as follows:

$$\begin{bmatrix} y_1^{(4)} \\ y_2^{(4)} \end{bmatrix} = b + A(x)A^{-1}(-b + v) = v \tag{2.205}$$

This completes the process of Dynamic extension!
In summary, if we can't make the matrix $A(x)$ invertible by taking derivatives of the output, we may *slow down* the appearance of an input by controlling its derivatives instead. This ensures all inputs appear at the same level of derivative of the output equation. To keep track of the lower derivatives of u_i as we control its higher derivatives, we append the lower derivatives to the state vector.

Chapter 3

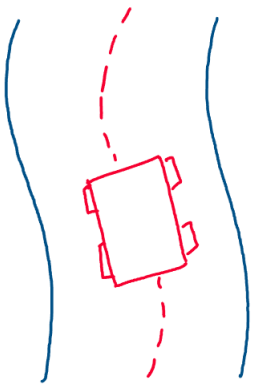
Nonholonomic Planning

Thus far, we've performed a review of some key concepts in the analysis and control of robotic systems. In the previous section, in studying some important classes of feedback controllers, we learned how to drive robotic systems to different desired states and to track different desired trajectories in their environment.
In this section, we'll think about some important design choices those desired trajectories should adhere to. We'll accomplish this through a thorough discussion of kinematic constraints, the fundamental constraints that exist on the motion of robotic systems. Let's get started!

3.1 Kinematic Constraints

Let's begin by reviewing what a trajectory actually is in the context of robotics. In robotics, trajectories are vector functions of time, $q(t) \in \mathbb{R}^n$, that contain positions we'd like our system to track.
If we have an initial state q_0 and a final state q_d that we'd like our trajectory to pass through, how can we determine what the function $q(t)$ should be?
In theory, we could set our desired trajectory $q(t)$ for our system to be *anything* we want! $q(t)$ could be a simple function that smoothly interpolates between our starting position, q_0, and our final position, q_d. It could also be something even simpler, such as a step function that jumps straight to a value of q_d.
We know from practice, however, that simply choosing an arbitrary path between two points won't produce optimal behavior in our robot! Remember that even if we choose a path that's simple or as short as possible, such as a straight line interpolation between two points, we *still* need our system to be able to track that path! A simple path *doesn't* guarantee that our system will be able to easily track the path. With this in mind, let's think about some important design considerations when coming up with a trajectory $q(t)$ we'd like our robot to follow.
Let's start by thinking about some simple spatial constraints we should have

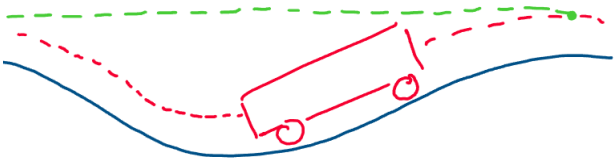
on our trajectory, and then work our way up to understanding constraints that make trajectories easier to follow. We can reason about some constraints we might want to enforce on our trajectories by thinking about the example of an autonomous car. Firstly, for a car trajectory, it's important to ensure our desired trajectories remain within the boundaries of the road!



Above: The car must stay within the boundaries of the road as it travels.

We can achieve this by setting a hard bounding constraint on the states our trajectory is able to pass through. If we come up with a trajectory from point q_0 to point q_d that goes *outside* of the boundaries of the road, we know that this trajectory won't be one we'd like to consider. What other constraints might be important to consider?

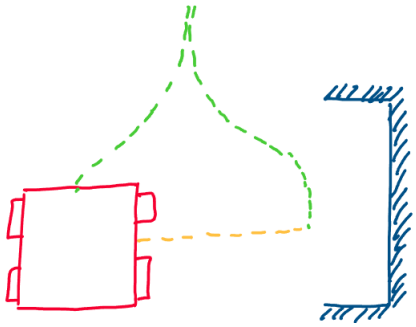
If our car is driving along a road, it's important that our desired trajectory $q(t)$ *follows the profile* of the road. We don't, for instance, want our car to have a desired trajectory that jumps through the air! We know our vehicle must be constrained to the surface of the road at all times.



Above: The trajectory must consider the geometry of the environment.

This way, we can ensure that our trajectory is one that is actually *dynamically feasible* for our system to track - our trajectory is something that actually respects the physical limits of the vehicle's motion. This brings us neatly to a last major constraint on our trajectories.

It's *critical* that whatever trajectory we generate is one that our car can actually follow! Even if we determine the quickest possible trajectory between two points, it's possible that our car won't be able to follow it! Consider, for instance, the problem of parallel parking.

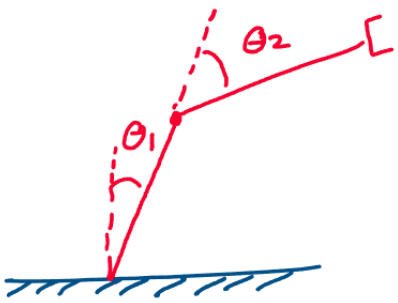


Above: Although the trajectory in yellow is the most direct, it does not respect the constraints on the vehicle.

Even though the shortest trajectory to move our car into the parked position is a straight, sideways line, drawn above in yellow, from our own experience with vehicles, we *know* there's no way our car can slide straight into the parking spot! Instead, we must plan an *alternate* trajectory that respects the possible motions of the vehicle, such as the parallel parking trajectory drawn above in green.
In this section, we'll focus in on the last two constraints we discussed above - that our trajectory must follow the profile of the road and respect the dynamic constraints of the vehicle. In generalizing our discussion of these constraints from cars to arbitrary systems, we'll look at constraints of these types through the lens of *kinematics*, the study of the structure of motion within systems.

3.1.1 Pfaffian Constraints

To develop some conventions for describing constraints, we'll find it useful to review an important method of describing the dynamics of systems: Lagrangian dynamics.
When determining a system's equations of motion using Lagrange's equations, recall that we first pick a set of **generalized coordinates**, which describe the positions of different components of the system. For example, in the robot arm:

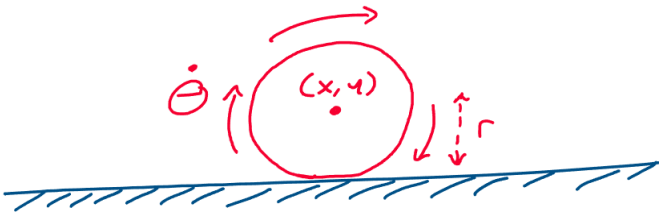


Above: We pick two generalized coordinates to describe the motion of the arm.

We could pick the generalized coordinates θ_1, θ_2 to describe the positions of different joints in the system. If we have a set of n generalized coordinates, which we can package for convenience in the vector $q = [q_1, q_2, ..., q_n]^T$, we may apply the Euler-Lagrange equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = F_{ext,i} \tag{3.1}$$

To each coordinate q_i to find the second order differential equations describing $\ddot{q}_i$. Can we use generalized coordinates to describe the constraints that exist on physical, robotic systems that may be described by Lagrangian dynamics?
To answer this question, we'll consider a few examples. First, let's think of a simple cylinder, which rolls without slipping along the surface of a table:



Above: A ball rolls without slipping across a flat surface.

When examining the effect of constraints on the system, it's often useful to begin by *removing* all of the constraints from the system and considering the generalized coordinates we would need to describe motion. In this case, we could remove the table, which gives us the free-floating and spinning cylinder:

Above: An unconstrained cylinder spins and flies through the air.

To describe the motion of this cylinder *without* any constraints, we could use two generalized coordinates (x, y) for the center of the cylinder, and one generalized coordinate, θ, for the rotational angle of the cylinder.
Let's now add the constraint of the table back in, and see if we can represent the constraints the table imposes on the cylinder in terms of the generalized coordinates we just found. Firstly, we know that the cylinder is constrained to the surface of the flat table. This gives us the constraint:

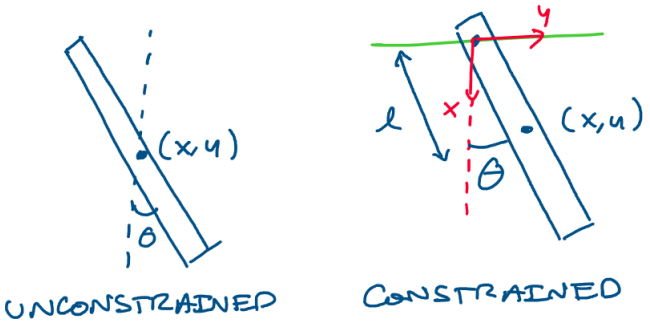
$$y = r \tag{3.2}$$

Since the y coordinate of the cylinder must be at the height given by the radius of the cylinder at all times. What other constraints do we have on the motion

of the cylinder? If the cylinder rolls without slipping along the table, we know that the velocity of the x coordinate of the center of mass, $\dot{x}$, is related to the angular velocity of the cylinder, $\dot{\theta}$, by the following equation:

$$\dot{x} = r\dot{\theta} \tag{3.3}$$

As we can see from this example, we may express the constraints on the motion of physical systems through expressions involving the generalized coordinates of the *unconstrained* system and their derivatives!
Let's do another, slightly more complex example, and see if this pattern holds up. Let's analyze the constraints on a rigid pendulum, which swings about a fixed pivot point in space.



Above: We need three coordinates, x, y, θ, to describe the unconstrained rigid bar, which translates and spins freely in space.

Once again, let's begin our analysis of the system by taking away the constraints. If we had an unconstrained rigid bar, we would need three different coordinates, x, y, θ, to completely describe the position and orientation of the bar! We could use x, y to describe the position of the center of mass, and θ to describe the angle of the bar with respect to the vertical.
As with the rolling cylinder, let's see if we can express the constraints imposed by the pendulum in terms of the generalized coordinates for the unconstrained system. Assuming an x axis that points vertically downwards, and a y axis that points to the right for convenience, we may use trigonometry to gain the following constraint equations:

$$x = l\cos\theta \tag{3.4}$$

$$y = l\sin\theta \tag{3.5}$$

Where l is the distance from the pivot point to the center of mass of the rigid bar. As with the cylinder, we observe that we can express the constraints on the motion of the constrained system *entirely* in terms of the generalized coordinates of the unconstrained system.
Now that we've explored two simple, physical examples of possible constraints on motion, let's see if we can work the sets of constraints above into a similar form. What common elements can we extract?

First, we notice that all of the constraints above are in terms of the generalized coordinates of a system and their derivatives. However, some constraints have no appearances of derivatives where others do!
To ensure we have a standardized way of representing a constraint that holds in *all* cases, we'll take the time derivative of the constraints that don't have any derivatives. This way, all of the constraint expressions will be in terms of the generalized coordinates and their first derivatives. For instance, for the rolling cylinder and the pendulum:

$$\dot{y} = 0 \tag{3.6}$$
$$\dot{x} = r\dot{\theta} \tag{3.7}$$
$$\dot{x} = -l\dot{\theta}\sin\theta \tag{3.8}$$
$$\dot{y} = l\dot{\theta}\cos\theta \tag{3.9}$$

Now, although of the constraints above are in terms of the generalized coordinates and their derivatives, the actual expressions for each constraint appear somewhat different! To standardize our method of representing constraints, we'll bring *everything* to the left hand side of the constraint expressions, so that we're left with a zero on the right hand side. For the constraints above:

$$\dot{y} = 0 \tag{3.10}$$
$$\dot{x} - r\dot{\theta} = 0 \tag{3.11}$$
$$\dot{x} + l\dot{\theta}\sin\theta = 0 \tag{3.12}$$
$$\dot{y} - l\dot{\theta}\cos\theta = 0 \tag{3.13}$$

Although all of the constraints now look somewhat more similar in form, we can still do a little bit better! We notice that the derivatives of the generalized coordinates in the expressions above may be *factored out* as vectors! We may factor out the vector $\dot{q}$ from each constraint to equivalently express the constraints as:

$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.14}$$
$$\begin{bmatrix} 1 & 0 & -r \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.15}$$
$$\begin{bmatrix} 1 & 0 & lsin\theta \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.16}$$
$$\begin{bmatrix} 0 & 1 & -l\cos\theta \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.17}$$

Where the first two constraints correspond to the rolling cylinder and the second two correspond to the pendulum.

Now, we've rewritten every kinematic constraint on the cylinder and pendulum systems as a product between a *row vector* that is a function of the generalized coordinates, and a *column vector* that contains the first derivatives of the generalized coordinates. This observation leads us to the following standard definition of a kinematic constraint.

Definition 27 ***Pfaffian Constraint***
Suppose we have a system with a vector of n generalized coordinates $q = [q_1, q_2, ..., q_n]^T$. A Pfaffian constraint on the generalized coordinates is a constraint of the form:

$$\omega_i(q)\dot{q} = 0 \tag{3.18}$$

Where $\omega_i(q) \in \mathbb{R}^n$ is a row vector that depends on the generalized coordinates and $\dot{q} \in \mathbb{R}^n$ is a column vector of derivatives of the generalized coordinates.

As we can see from the definition of a Pfaffian constraint, our rewritten constraints for the cylinder and the ball were *all* Pfaffian constraints, as each had a row vector that was a function of the generalized coordinates multiplied by a vector containing the derivatives of the generalized coordinates. Pfaffian constraints are a large group that may be used to express the kinematic constraints on many physical systems.
Let's think a little bit deeper about the types of relationships Pfaffian constraints impose on systems through analyzing a simple, three-dimensional example. Note that going forward, we'll frequently refer to the first time derivatives of generalized coordinates as the *velocities* of the generalized coordinates.
Suppose we have a Pfaffian constraint $\omega_i(q)\dot{q}$, where $q \in \mathbb{R}^3$. Then, the Pfaffian constraint may be expressed:

$$\omega_i(q)\dot{q} = \begin{bmatrix}\omega_{i1}(q) & \omega_{i2}(q) & \omega_{i3}(q)\end{bmatrix} \begin{bmatrix}\dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3\end{bmatrix} = 0 \tag{3.19}$$

$$\omega_{i1}(q)\dot{q}_1 + \omega_{i2}(q)\dot{q}_2 + \omega_{i3}(q)\dot{q}_3 = 0 \tag{3.20}$$

We observe that the j^{th} entry of the $\omega_i(q)$ row vector, $\omega_{ij}(q)$, tells us how the j^{th} generalized coordinate will be affected by the constraint. For instance, if $\omega_{i2}(q) = 0$, then we know that the velocity of the second generalized coordinate, $\dot{q}_2$, will not be impacted by the constraint. Overall, from the relationship:

$$\omega_{i1}(q)\dot{q}_1 + \omega_{i2}(q)\dot{q}_2 + \omega_{i3}(q)\dot{q}_3 = 0 \tag{3.21}$$

We observe that a Pfaffian constraint *explicitly* imposes a constraint on the velocities $\dot{q}$ of the generalized coordinates at each position q.
At this point in our development of constraints, we have to ask an *extremely*

important question. In addition to explicitly constraining the *velocities* of a system's generalized coordinates, does a Pfaffian constraint explicitly constrain the *positions* of a system's generalized coordinates? Let's reason about the answer to this question through some examples.
Let's start by turning our attention back to the rigid pendulum. We know that the Pfaffian constraints on the velocities of the generalized coordinates of the pendulum:

$$\begin{bmatrix} 1 & 0 & lsin\theta \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.22}$$

$$\begin{bmatrix} 0 & 1 & -l\cos\theta \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.23}$$

Are *equivalent* to the following explicit constraints on the positions of the generalized coordinates of the pendulum:

$$x - l\cos\theta = 0 \tag{3.24}$$

$$y - l\sin\theta = 0 \tag{3.25}$$

In this case, then, it appears that the Pfaffian constraints *do* explicitly constrain the positions of the generalized coordinates, as they are equivalent to a direct relationship between x, y, θ.
What about in another case, such as the parallel-parking car? In this case, we may prove that we can still represent the constraints on the car's motion in Pfaffian form, $\omega_i(q)\dot{q} = 0$, where q is a vector of generalized coordinates needed to describe the car's motion.
From our experience with cars, however, we know that despite the constraints on its motion, we can *still* move a car to any position we like on the road! Thus, in the case of a car, the Pfaffian constraint *does not* seem to explicitly constrain the positions of the generalized coordinates. Rather, the Pfaffian constraint *just* constrains the velocities of the generalized coordinates.
How can we formally distinguish between these two different types of Pfaffian constraints - one which constrains positions and one which doesn't? The answer to this question lies in something called *integrability*.

3.1.2 Holonomic & Nonholonomic Constraints

In this section, we'll work on classifying both single Pfaffian constraints and systems of Pfaffian constraints. Let's begin with the simpler case of a single Pfaffian constraint.

A Single Pfaffian Constraint

Let's formally classify the two types of constraints we discussed in our physical examples above. If a Pfaffian constraint $\omega_i(q)\dot{q} = 0$ on the velocities of a system's

generalized coordinates is *equivalent* to a constraint directly on the positions of the generalized coordinates, the Pfaffian constraint is said to be **integrable**.
We can reason about this choice of word through the following mnemonic - if we *integrate* velocity in time, we get *position* back out! Likewise, if a Pfaffian constraint on the velocities of generalized coordinates is *integrable*, it's equivalent to a constraint directly on the *positions* of generalized coordinates.
Let's discuss the concept of integrability in further mathematical detail, and state the definition more precisely.

Definition 28 ***Integrable Constraint***
A Pfaffian constraint $\omega_i(q)\dot{q} = 0$, where $q \in \mathbb{R}^n$ is a vector of generalized coordinates, is said to be integrable if there exists a function $h(q)$ such that:

$$\omega_i(q)\dot{q} = 0 \rightarrow h(q) = 0 \tag{3.26}$$
$$h(q) = 0 \rightarrow \omega_i(q)\dot{q} = 0 \tag{3.27}$$

In other words, a Pfaffian constraint is said to be integrable if all $q(t)$ satisfying the Pfaffian constraint satisfy the algebraic constraint $h(q) = 0$, and all $q(t)$ satisfying the algebraic constraint $h(q) = 0$ also satisfy the Pfaffian constraint $\omega_i(q)\dot{q} = 0$. This means that the set of all trajectories $q(t)$ satisfying the two constraints are the same, which makes the constraints *equivalent*.
Let's reason about integrability in terms of our physical examples from above. We know that the Pfaffian constraints on the velocities of the generalized coordinates:

$$\begin{bmatrix} 1 & 0 & lsin\theta \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.28}$$

$$\begin{bmatrix} 0 & 1 & -l\cos\theta \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.29}$$

Are *equivalent* to the two constraints $x - l\cos\theta = 0, y - l\sin\theta = 0$ on the *positions* of generalized coordinates. In other words, these two Pfaffian constraints are integrable, as they're both equivalent to expressions of the form $h(q) = 0$.
Mathematically, how can we tell if a constraint is integrable or not? Suppose we have a Pfaffian constraint $\omega(q)\dot{q} = 0$, which we'd like to show is equivalent to an algebraic constraint $h(q) = 0$.
What properties do $h(q), \omega(q)$ need to give equivalent constraints? Since the Pfaffian constraint is in terms of $\dot{q}$, let's begin by taking the time derivative of $h(q)$, to turn it into an expression involving velocities. Differentiating using the chain rule:

$$\dot{h}(q) = \frac{\partial h}{\partial q}\dot{q} = 0 \tag{3.30}$$

Where $\frac{\partial h}{\partial q}$ is a row vector containing the partial derivatives of h with respect to the generalized coordinates q_i. Comparing this to the expression for a Pfaffian constraint:

$$\omega(q)\dot{q} = 0 \tag{3.31}$$

We see that once again, we have a row vector multiplied by the velocities of the generalized coordinates. Thus, we conclude that since both expressions involve a row vector multiplied by $\dot{q}$, and are equal to zero, they simply must be equivalent up to a scale factor!
In other words, for $\omega(q)\dot{q} = 0$ to be equivalent to $h(q) = 0$, there must exist a scalar function $\alpha(q)$ such that:

$$\alpha(q)\omega(q)\dot{q} = \frac{\partial h}{\partial q}\dot{q} = 0 \tag{3.32}$$

Since all that we're doing is multiplying a zero by a scale factor function. If we can find an $\alpha(q)$ and an $h(q)$ to satisfy this property, we *know* that the Pfaffian constraint will be equivalent to a constraint $h(q) = 0$ and will therefore be integrable. In this case, the scalar function $\alpha(q)$ that achieves this property is called an **integrating factor**.
Note that although finding an integrating factor and a function $h(q)$ is sufficient to prove integrability of a single Pfaffian constraint, the process of finding an integrating factor is *not* always an easy one! Additionally, when we look at systems of Pfaffian constraints, finding an integrating factor will no longer generally be sufficient to prove the integrability of a system of constraints.

Systems of Pfaffian Constraints

Now that we've considered the case of a single Pfaffian constraint in reasonable mathematical detail, let's think about what happens when we have a *system* of Pfaffian constraints! In this case, we'll no longer look at one Pfaffian constraint of the form $\omega_i(q)\dot{q} = 0$, but will rather explore the properties of a set of k independent Pfaffian constraints:

$$\{\omega_1(q)\dot{q} = 0, ..., \omega_k(q)\dot{q} = 0\} \tag{3.33}$$

Note that here, the word *independent* simply means that none of the constraints are redundant with one another, and each imposes a new condition on the motion of the system.
Let's develop some terminology for discussing the behavior of systems of constraints. First, let's consider the case where all k constraints are integrable.

Definition 29 ***Holonomic Constraint***
A set of k Pfaffian constraints $\{\omega_1(q)\dot{q} = 0, ..., \omega_k(q)\dot{q} = 0\}$ is said to be holonomic if every constraint $\omega_i(q)\dot{q} = 0$ is integrable.

In other words, for a system of Pfaffian constraints to be holonomic, each $\omega_i(q)\dot{q} = 0$ must be equivalent to an algebraic constraint, $h_i(q) = 0$, on the generalized coordinates of the system.
Now that we've defined the case where *all* k constraints are integrable, let's consider the case where the constraints are not integrable.

Definition 30 ***Nonholonomic Constraint***
A set of k Pfaffian constraints $\{\omega_1(q)\dot{q} = 0, ..., \omega_k(q)\dot{q} = 0\}$ *is said to be completely nonholonomic if none of the constraints* $\omega_i(q)\dot{q} = 0$ *are integrable.*
If some of the k Pfaffian constraints are integrable and some are not, the set of constraints is said to be partially nonholonomic.

For convenience, we'll generally refer to both completely and partially nonholonomic systems simply as being nonholonomic.
Can we mathematically prove a system of constraints is holonomic or nonholonomic in the same way we can prove a single constraint is integrable?
Unfortunately, it turns out that mathematically proving a system of Pfaffian constraints is holonomic or nonholonomic is an *extremely challenging* task - much more so than the case of a single Pfaffian constraint. To formally prove that a system of constraints is holonomic or not, we must appeal to a field of mathematics known as *differential geometry.*
If you're interested in learning more about the formal verification of holonomic and nonholonomic constraints, you're encouraged to read chapter 7 of a *Mathematical Introduction to Robotic Manipulation* by Murray, Li, and Sastry or chapter 4 of *Analytical Dynamics of Discrete Systems* by Rosenberg for more information.
Although the *formal* verification of general holonomic versus nonholonomic systems is quite a nontrivial task, it turns out that in many cases, we can use our physical *intuition* to determine if constraints are holonomic or not.
To reason about the holonomy of constraints in an intuitive physical manner, we use the following procedure. Note that this procedure *is* something that we can verify formally for correctness!

Proposition 5 ***Determining Holonomy of Physical Constraints***
Consider an unconstrained physical system described by n generalized coordinates, $q_1, q_2, ..., q_n$*. Suppose we apply k independent Pfaffian constraints to the system. To determine if the constraints are holonomic or nonholonomic, we may follow the procedure outlined below:*

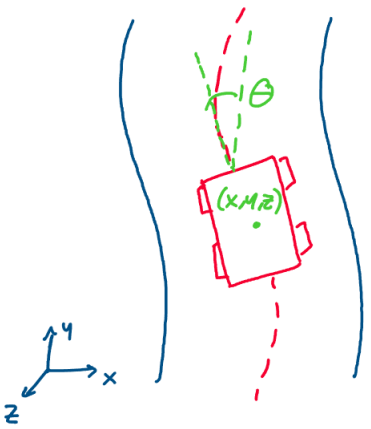
1. *Begin with the unconstrained physical system, where we need all n generalized coordinated* $q_1, q_2, ..., q_n$ *to describe the motion of the system.*

2. *Add the first Pfaffian constraint $\omega_i(q)\dot{q} = 0$ to the system and reason about the motion of the resulting constrained system.*
3. *If the constraint restricts the positions the generalized coordinates can move to and reduces the number of generalized coordinates we need by one, the constraint is integrable. Otherwise, the constraint is non-integrable.*
4. *Repeat steps 2 and 3 for the remaining Pfaffian constraints on the system to determine if the set of constraints is holonomic or nonholonomic.*

This procedure brings up an important point about describing the dynamics of constrained systems. Since a single integrable constraint *reduces* the number of generalized coordinates required to describe the system's motion by one, we may use the following formula to determine the number of generalized coordinates needed to describe the motion of constrained systems.

$$\text{Number of G.C.s} = \text{Number of Unconstrained G.C.s} \tag{3.34}$$
$$- \text{Number of Integrable Constraints} \tag{3.35}$$

By subtracting the number of (independent) integrable constraints from the number of generalized coordinates needed to describe the unconstrained system, we find the number of generalized coordinates we need to describe the constrained system.
Since each integrable constraint is equivalent to an algebraic constraint of the form $h_i(q) = 0$, each constraint allows us to solve for one of the generalized coordinates in terms of the others, thus making one generalized coordinate *redundant*. This decreases the number of generalized coordinates we need to completely describe the motion of the system.
Let's apply the procedure above to determine the nature of the constraints on a simple car, which moves around in the flat plane at $z = 0$.



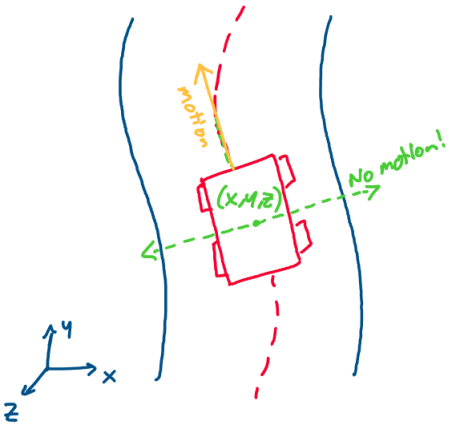
Above: A car travels in the x, y plane with a heading angle θ.

This car has two main constraints on its motion: firstly, it must remain in the plane, at $z = 0$, and secondly, it cannot slide freely side to side. Let's follow the procedure to determine if these constraints are integrable or not!

First, we consider the unconstrained system. To describe the unconstrained system, we can use four generalized coordinates: x, y, z, θ, where x, y, z are the coordinates of the center of mass of the car, and θ is the car's heading angle with respect to the vertical.

Let's consider the first constraint on the system, that the car is constrained to move around in the flat plane at $z = 0$. Clearly, this constraint *restricts* the positions the generalized coordiantes are able to move to, as it directly sets the z coordinate of the car to zero at all times. Additionally, since it sets $z = 0$ for all time, it eliminates the need for a z generalized coordinate, as we always know what z will be. By our criteria above, this constraint is integrable.

Now, we consider the next constraint - that the car can't slide side to side. We know from experience with vehicles in the real world that despite this constraint, we can still move our car anywhere we want in the plane! Thus, this second constraint *doesn't* enforce any restrictions on the positions the system is able to travel to, and is therefore non-integrable.



Above: Although the car is constrained to not slide side to side, we can still steer it anywhere in space.

Since we have one integrable and one non-integrable constraint, we conclude that the system of constraints is partially nonholonomic. We can use this sort of physical intuition to reason about the holonomy of constraints on *many* physical systems.

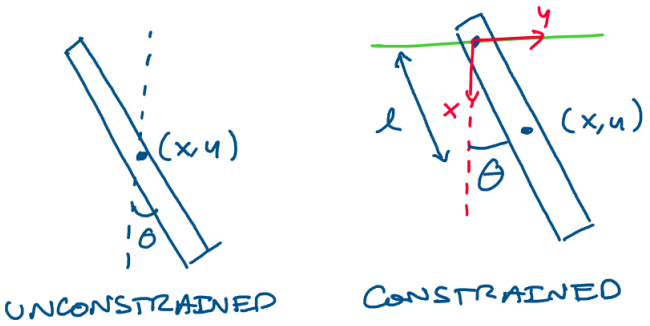
Let's try another example: the rigid pendulum. Starting with the unconstrained system, we recall that to describe the motion of an unconstrained rigid bar, we need three generalized coordinates, (x, y, θ). When we add the first pendulum constraint to the bar:

$$x - l\cos\theta = 0 \tag{3.36}$$

We see that the values of x reachable by the bar are restricted! Further, we can solve for x in terms of another generalized coordinate, θ, which eliminates the need for the x generalized coordinate. This makes this constraint integrable.
We may similarly treat the second constraint on the pendulum:

$$y - l\sin\theta = 0 \tag{3.37}$$

We see that we can also solve for y in terms of θ, which eliminates the need for the y generalized coordinate. Further, the values of y are restricted. This makes the second constraint integrable as well.
Thus, when we turn the unconstrained rigid bar into a pendulum by adding two integrable constraints, we only need $3 - 2 = 1$ generalized coordinates, for example θ, to completely describe the motion of the bar.



Above: Two integrable constraints means we require two fewer generalized coordinates than the unconstrained system.

Since all of the constraints on the pendulum are integrable, we conclude that the set of pendulum constraints are holonomic.

3.1.3 Equivalent Control Systems

Thus far, we've constrained our discussion to focus on the structure of constraints on robotic and physical systems. Let's bring our development of the theory of constraints back to the realm of planning and control systems, and see how we can exploit the fundamental structure of constraints to control various robotic systems.
Let's begin with the same system of k independent Pfaffian constraints, $\omega_1(q)\dot{q} = 0, ..., \omega_k(q)\dot{q} = 0$, where $q \in \mathbb{R}^n$ is a vector of n generalized coordinates. Let's rewrite the system of k constraints in matrix form, and see what conclusions we can make about the constraints using linear algebra. We know:

$$\omega_1(q)\dot{q} = 0 \tag{3.38}$$
$$\omega_2(q)\dot{q} = 0 \tag{3.39}$$
$$\vdots \tag{3.40}$$
$$\omega_k(q)\dot{q} = 0 \tag{3.41}$$

Where each $\omega_i(q)$ is an n dimensional row vector. Thus, if we place each $\omega_i(q)$ in a row of a matrix, we may rewrite this system of constraints as:

$$\begin{bmatrix} - & \omega_1(q) & - \\ & \vdots & \\ - & \omega_k(q) & - \end{bmatrix} \dot{q} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.42}$$

If we name the matrix of $\omega_i(q)$ row vectors $A(q) \in \mathbb{R}^{k \times n}$, we see that the system of Pfaffian constraints is of the form:

$$A(q)\dot{q} = 0 \tag{3.43}$$

Let's use our knowledge of linear algebra to interpret the conditions this system of constraints enforces on $\dot{q}$. The expression $A(q)\dot{q} = 0$ tells us the incredibly important property that if $q(t)$ is a trajectory that satisfies all of the constraints on the system, then $\dot{q}(t)$ *must* be in the null space of $A(q)$.
If $\dot{q}(t)$ is *not* in the null space of $A(q)$, it does not satisfy the Pfaffian constraints on the system, and $q(t)$ is therefore *not* a trajectory that respects the laws governing the motion of the system.
Recognizing that the null space of $A(q)$ tells us *all* of the allowable values of $\dot{q}$, we know that if we can find a basis for the null space of $A(q)$, we'll be able to identify *every* $\dot{q}(t)$, and therefore every trajectory $q(t)$, that respects the constraints on the system's motion.
Let's think about what the null space of $A(q)$ might look like. If we have n generalized coordinates for the unconstrained system, and k independent Pfaffian constraints on the velocities of the generalized coordinates, we may prove that the null space of the matrix $A(q)$ is:

$$m = n - k \tag{3.44}$$

Dimensional. In other words, there is a set of m linearly independent vectors:

$$g_1(q), g_2(q), ..., g_m(q) \tag{3.45}$$

Such that $A(q)g_i(q) = 0$. Notice that because $A(q)$ is a function of q, the basis vectors for the null space of $A(q)$ are *also* a function of a q. Since each basis vector $g_i(q)$ takes in a vector q and returns *another* vector $g_i(q)$, we note that the basis vectors of the null space of $A(q)$ are not simply vectors, but are rather *vector fields*.
From linear algebra, we know that any linear combination of the basis vectors of the null space of $A(q)$ will *still* be in the null space of $A(q)$. This means that any vector of the form:

$$u_1 g_1(q) + ... + u_m g_m(q) \tag{3.46}$$

Where each $u_i \in \mathbb{R}$ is an arbitrary scalar, *still* must be in the null space of $A(q)$!

$$A(q)(u_1 g_1(q) + ... + u_m g_m(q)) = 0 \tag{3.47}$$

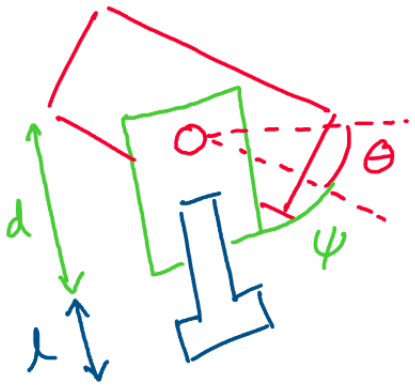
Since all valid values of $\dot{q}$ are in the null space of $A(q)$, by the definition of a basis, *all* $\dot{q}$ in the null space of $A(q)$ may be represented as a linear combination of the basis vectors $g_1, ..., g_m$. Thus, we know *every* $\dot{q}$ that satisfies the constraints on the system may be represented in the form:

$$\dot{q} = u_1 g_1(q) + ... + u_m g_m(q) \tag{3.48}$$

Miraculously, we now have a system that perfectly describes all possible values of $\dot{q}(t), q(t)$ of the system. What's more, this representation has a set of m scalars that we can change arbitrarily! Because of this, we say that the system $\dot{q} = u_1 g_1(q) + ... + u_m g_m(q)$ is an **equivalent control system** for the system, since by choosing $u_1, ..., u_m$, we can find the value of $\dot{q}$.
These equivalent control systems act *just* like the control affine systems we discussed earlier in the course, which had the form:

$$\dot{x} = f(x) + g(x)u \tag{3.49}$$

An equivalent control system of the form above is simply a control affine system where $f(x) = 0$ and the state vector $x = q$ is a vector of generalized coordinates. Because of this equivalence, we can use all of the methods of feedback control we developed previously to drive our constrained equivalent control systems to the positions we want.
Let's consider a practical example of an equivalent control system. Famously, Marc Raibert, the founder of the robotics company Boston Dynamics, employed equivalent control systems in the design of a hopping robot called *Raibert's hopper*. Let's set up the equivalent control system for Raibert's hopper:



Above: A simplified model of Raibert's hopper.

Note that the length of extension of the leg, l, can be changed to propel the system into the air, while the angle ψ can be modified to change the angle of the jump into the air. From conservation of angular momentum, the following constraint can be shown to exist on the hopper as it travels through the air:

$$(I + m(l+d)^2)\dot{\theta} + m(l+d)^2\dot{\psi} = 0 \tag{3.50}$$

Where I, m are the moment of inertia and mass of the body, respectively. Choosing a vector of generalized coordinates $q = [\psi, l, \theta]^T$, we may rewrite this constraint in Pfaffian form as:

$$\omega(q)\dot{q} = \begin{bmatrix} m(l+d)^2 & 0 & I + m(l+d)^2 \end{bmatrix} \begin{bmatrix} \dot{\psi} \\ \dot{l} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.51}$$

Let's try to find a basis for the null space of the constraint $\omega(q)$! Since we have three generalized coordinates, ψ, l, θ, and one Pfaffian constraint, we conclude that there should be $3 - 1 = 2$ basis vectors for the null space of the constraint. Looking at the constraint, we can pick the following for the first basis vector:

$$g_1(q) = \begin{bmatrix} 1 \\ 0 \\ -\frac{m(l+d)^2}{I+m(l+d)^2} \end{bmatrix} \tag{3.52}$$

If we multiply the constraint by this choice of $g_1(q)$, the first and third entries in the product $\omega(q)g_1(q)$ will cancel out, leaving us with zero. Noticing that the middle entry of $\omega(q) = 0$, we can pick the following for our second basis vector:

$$g_2(q) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.53}$$

Thus, we may use the following equivalent control system to control the hopping robot as it flies through the air:

$$\dot{q} = \begin{bmatrix} 1 \\ 0 \\ -\frac{m(l+d)^2}{I+m(l+d)^2} \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} u_1 \tag{3.54}$$

$$\dot{q} = g_1(q)u_1 + g_2(q)u_2 \tag{3.55}$$

We know that by adjusting u_1, u_2, we'll always end up with a $\dot{q}$ that respects the constraints on the system dynamics, since all linear combinations of g_1, g_2 will still be in the null space of the constraint.
Notice that the basis we pick for the null space is not unique, but some bases, such as that which we picked above, can be more convenient than others when designing controllers.

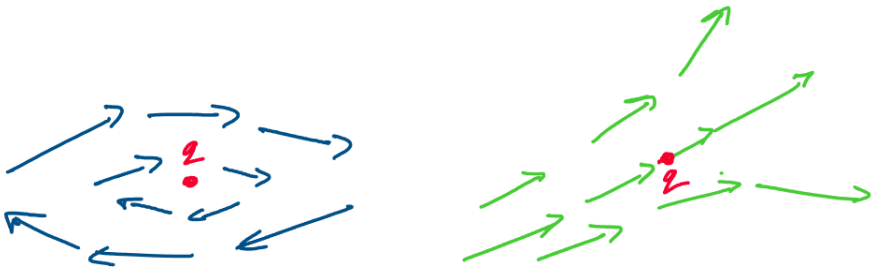
3.1.4 Lie Brackets & Controllability

We've now successfully converted our kinematic constraints into an equivalent control system, $\dot{q} = g_1(q)u_1 + ... + g_m(q)u_m$ that we may use to control our constrained system. Now, the question we ask is - can we drive the system from any starting state q_0 to any final state q_d in a finite amount of time while

respecting the constraints on the system's motion?
Let's think about how we can answer the question of being able to drive the system from any initial set of generalized coordinates $q \in \mathbb{R}^n$ to any final set of generalized coordinates $q_d \in \mathbb{R}^n$. What does our constraint dynamics:

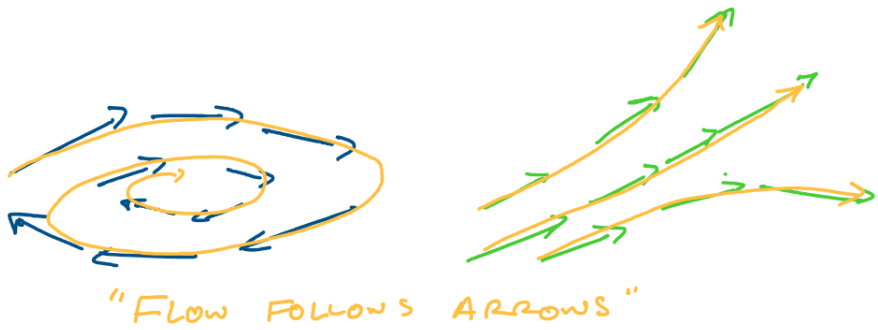
$$\dot{q} = g_1(q)u_1 + ... + g_m(q)u_m \tag{3.56}$$

Tell us about the allowable motions of our system? From looking at the expression, we know that all allowable *directions* of motion, $\dot{q}$, must be along the vector fields $g_1, g_2, ..., g_m$. Thus, at each point q in space, the vector fields $g_1, ..., g_m$ tell us the directions of motion we're able to travel in.



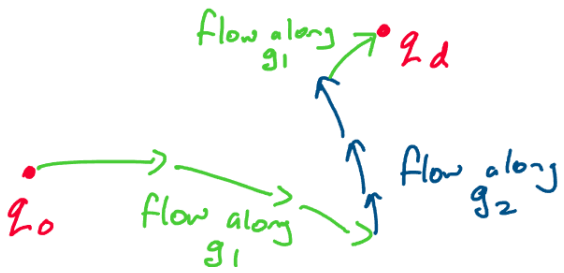
Above: If $\dot{q} = g_1(q)u_1 + g_2(q)u_2$, at any point q, we may travel along either the g_1 or g_2 vector field, or any combination of the two.

Since the vector fields $g_1, ..., g_m$ tell us the directions our trajectory can "point" at any q in space, by following combinations of paths along the vectors of the vector fields, we can trace out the *allowable trajectories* $q(t)$ of the system! We refer to these paths along the vector fields as **flows**.



Above: The allowable values of $q(t)$ are found by taking linear combinations of flows along the vector fields.

By thinking about the motions of our system as being movements along these vector fields at all times, to determine if we can move from a state q_0 to a state q_d, we can try to determine if we can split up the overall motion as a set of small motions along the vector fields $g_1, ..., g_m$.



Above: Can we split up the motion from q_0 to q_d by travelling along the vector fields $g_1, ..., g_m$?

If we can *successfully* travel from q_0 to q_d by moving along these vector fields, then we can reach our desired state in finite time while respecting the constraints on the system's motion.

Now that we've thought conceptually about how to determine our ability to reach different states, let's develop a systematic *mathematical* procedure! We'd now like to mathematically determine if we can reach an arbitrary state $q_d \in \mathbb{R}^n$ from an initial state q_0 in finite time by travelling along the m vector fields $g_1, ..., g_m$. To develop a procedure for determining such an ability, we'll need to develop a few more mathematical tools!

Lie Brackets & Lie Algebras

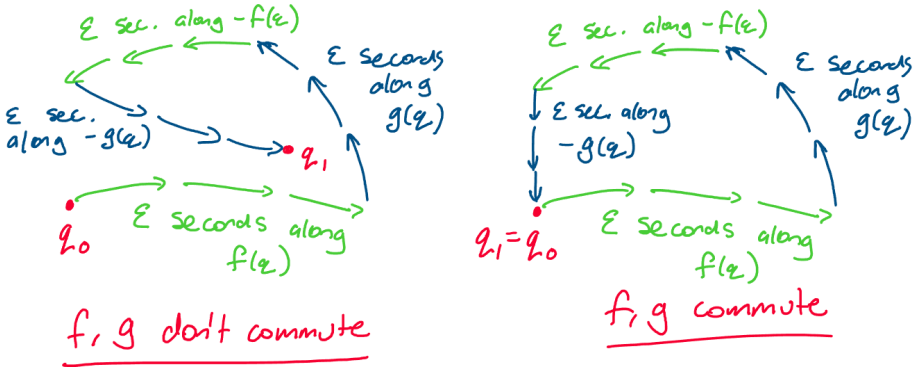
Before we begin discussing mathematical specifics, let's take a moment to appreciate the challenge of the task before us! By travelling along m linearly independent vector fields $g_1, ..., g_m$, we'd like to be able to reach arbitrary locations in $\mathbb{R}^n$, where $m < n$.

To achieve this, we'll need to show that by composing motions along the m vector fields, we can increase the directions we can travel in from the m directions along $g_1, ..., g_m$ to a full range of n directions. This will enable us to travel anywhere in space at a given point q.

Let's think about the idea of composing motions along vector fields with a simple example. Suppose we have two vector fields $f(q), g(q) \in \mathbb{R}^n$, where $q \in \mathbb{R}^n$. Let's try out a simple composition of motions along these vector fields, and see if we can generate any new directions of motion! If we *can* generate a new direction of motion, perhaps we'll be able to reach a large set of positions in space by travelling along these vector fields!

Consider the following symmetric motion: first, starting from q_0, we'll travel ε seconds along the vector field $f(q)$. Next, we'll travel ε seconds along the vector field $g(q)$. Following this, we'll travel ε seconds in the direction of $-f(q)$. Finally, we'll travel ε seconds in the direction of $-g(q)$.

This simple, symmetric motion, known as a **Lie bracket motion**, helps us tell if we can generate a *new* direction of motion by composing motions along the two vector fields $f(q), g(q)$.



Above: A Lie bracket motion along two vector fields. In the left example, f, g do not commute. In the right, they do.

If after completing this symmetric motion, we end up at the *same* location as where we started, q_0, we know that the symmetric motion along the two vector fields *didn't* allow us to reach a new location! In this case, the vector fields f and g are said to **commute**.
On the other hand, if after completing this motion, we end up at a *new* location $q_1 \neq q_0$, we know that we can reach a new location by performing this motion along the vector fields. In this case, f and g do not commute.
Let's introduce a little bit of mathematics to compute whether this symmetric motion along the vector fields takes us back to our starting location. By assuming that the time we travel along the vector fields, ε, is small, we may use a second order Taylor approximation to derive the following famous expression.

Definition 31 ***Lie Bracket***
The Lie bracket of two vector fields $f(q), g(q) \in \mathbb{R}^n$ is defined:

$$[f, g](q) = \frac{\partial g}{\partial q} f(q) - \frac{\partial f}{\partial q} g(q) \tag{3.57}$$

Where $[f, g](q)$ is another vector field in $\mathbb{R}^n$. If $[f, g](q) = 0$, then f, g are said to commute.

Moving forward, we'll use this surprisingly simple expression to help us determine if we can generate new directions of motion by composing motions along f and g. If $[f, g](q) \neq 0$, thinking back to our definition of a Lie bracket motion, we might be able to use $[f, g](q)$ itself as a new direction of motion!
Let's perform a simple example of a Lie bracket computation to check our understanding. Suppose we want to find the Lie bracket of the following two vector

fields, which are functions of a vector $q = [q_1, q_2, q_3]^T$:

$$g_1(q) = \begin{bmatrix} 1 \\ 0 \\ q_2 \end{bmatrix}, \; g_2(q) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.58}$$

By the definition of a Lie bracket, we must calculate:

$$[g_1, g_2](q) = \frac{\partial g_2}{\partial q} g_1(q) - \frac{\partial g_1}{\partial q} g_2(q) \tag{3.59}$$

Let's start by calculating the partial derivatives, $\frac{\partial g_2}{\partial q}$ and $\frac{\partial g_1}{\partial q}$. Since we're taking the partial derivatives of vectors with respect to vectors, these two expressions will both be *matrices*.

$$\frac{\partial g_2}{\partial q} = \begin{bmatrix} \frac{\partial g_{21}}{\partial q_1} & \frac{\partial g_{21}}{\partial q_2} & \frac{\partial g_{21}}{\partial q_3} \\ \frac{\partial g_{22}}{\partial q_1} & \frac{\partial g_{22}}{\partial q_2} & \frac{\partial g_{22}}{\partial q_3} \\ \frac{\partial g_{23}}{\partial q_1} & \frac{\partial g_{23}}{\partial q_2} & \frac{\partial g_{23}}{\partial q_3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.60}$$

$$\frac{\partial g_1}{\partial q} = \begin{bmatrix} \frac{\partial g_{11}}{\partial q_1} & \frac{\partial g_{11}}{\partial q_2} & \frac{\partial g_{11}}{\partial q_3} \\ \frac{\partial g_{12}}{\partial q_1} & \frac{\partial g_{12}}{\partial q_2} & \frac{\partial g_{12}}{\partial q_3} \\ \frac{\partial g_{13}}{\partial q_1} & \frac{\partial g_{13}}{\partial q_2} & \frac{\partial g_{13}}{\partial q_3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.61}$$

Now that we have these two matrices of partial derivatives, we simply multiply the matrices by the vector fields to complete our computation of the Lie bracket:

$$[g_1, g_2](q) = \frac{\partial g_2}{\partial q} g_1(q) - \frac{\partial g_1}{\partial q} g_2(q) \tag{3.62}$$

$$[g_1, g_2](q) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ q_2 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{3.63}$$

Since the result of the Lie bracket computation was *nonzero*, we conclude that the two vector fields g_1, g_2 *do not* commute! Thus, they have the potential to generate new directions of motion! This idea of generating new directions of motion by using Lie brackets bring us to the following concept.

Definition 32 ***Lie Algebra***
The Lie algebra of a set of vector fields $g_1, g_2, ..., g_m \in \mathbb{R}^n$, denoted $\mathcal{L}(g_1, ..., g_m)$ is the span of the vector fields and their Lie brackets.

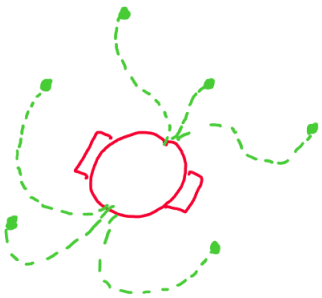
$$\mathcal{L}(g_1, ..., g_m) = span\{g_1, ..., g_m, [g_1, g_2], ..., [g_{m-1}, g_m], [g_1, [g_1, g_2]], ...\} \tag{3.64}$$

We define the **dimension** of a Lie algebra to be the number of linearly independent vector fields we can generate from the vector fields $g_1, ..., g_m$ and their

Lie brackets.
Notice how in addition to computing Lie brackets such $[g_1, g_2]$, we also compute further *nested* Lie brackets such as $[g_1, [g_1, g_2]]$, which are called higher order Lie brackets. Since each Lie bracket simply returns a vector field, the computations required to calculate these higher order, nested Lie brackets involve reapplying the same basic Lie bracket definition.
Since the Lie algebra of a set of vector fields represents the span of all of the vector fields and their Lie brackets, it helps us identify the possible directions of travel at each point in space by moving along the vector fields $g_1, ..., g_m$, and their compositions.
Equipped with these tools, we may return to the problem of driving our system to arbitrary states!

Controllability

The idea of being able to drive a system from any starting state to any desired state in finite time is a key concept in control known as *controllability*, which we introduced briefly in our exploration of linear control.
When discussing linear control systems, recall that we defined a system to be controllable at q_0 if there existed an input $u(t)$ to drive the system from its starting state q_0 to a final state q_f within some finite time interval $[0, T]$.



Above: Can we drive a turtlebot to any of its surrounding states in finite time?

In this section, we'll discuss controllability at a more advanced level, and will extend our characterization of controllability to the set of nonlinear, equivalent control systems of the form:

$$\dot{q} = g_1(q)u_1 + ... + g_m(q)u_m \tag{3.65}$$

When studying controllability, since it's important to have an understanding of the states our system is able to reach within certain amounts of time, we make the following definition.

Definition 33 ***Reachable States***
Let $V \subseteq \mathbb{R}^n$ be a region of space inside $\mathbb{R}^n$. The reachable states of a system

$\dot{q} = g_1(q)u_1 + ... + g_m(q)u_m$ in time t from an initial state $q_0 \in V$, denoted:

$$\mathcal{R}^V(q_0, t) \tag{3.66}$$

Is defined to be the set of all states that may be reached by the system at time t, starting from q_0 and staying within the set V.

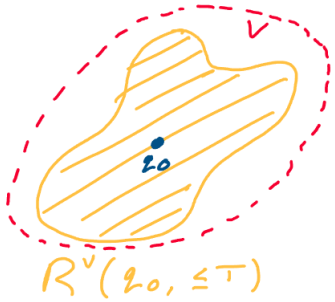
The set of reachable states $\mathcal{R}^V(q_0, t)$ tells us which states we can reach at *exactly* time t, starting from q_0. What about the states we can reach in all times from 0 to T starting from q_0 and staying within the region V?

Definition 34 ***Reach Set***
Let $T \in \mathbb{R}^+$ and $V \subseteq \mathbb{R}^n$. The reach set of a nonlinear system $\dot{q} = g_1(q)u_1 + ... + g_m(q)u_m$ is defined to be the union of all reachable states of the system across times $t = 0$ to $t = T$. It is denoted:

$$\mathcal{R}^V(q_0, \leq T) = \bigcup_{0 \leq t \leq T} \mathcal{R}^V(q_0, t) \tag{3.67}$$

Where $\mathcal{R}^V(q_0, t)$ is the set of reachable states starting from q_0 within a time t.

Note that in the definition above $\bigcup_{0 \leq t \leq T}$ represents the *union*, or combined collection, of all of the sets $\mathcal{R}^V(q_0, t)$, where t ranges from 0 to T.
It's important to note that the *only* difference between the set of reachable states at time t from q_0 and the reach set from q_0 is that the reach set contains all of the states reachable for all times t *up to* a time T, rather than just for a single time t.



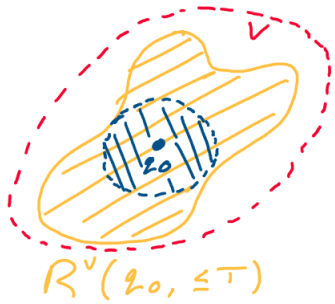
Above: The reach set from q_0 in time T is the set of all states we can reach from q_0 within a time T while staying within a region V.

Using the concept of a reach set, let's formally state the definition of an important type of controllability. First, we'll state a mathematical definition and then give a more intuitive interpretation.

Definition 35 ***Small Time Local Controllability***
A system $\dot{q} = g_1(q)u_1 + ... + g_m(q)u_m$ is small time locally controllable at $q_0 \in \mathbb{R}^n$ if its reach set around q_0 within time T, $\mathcal{R}^V(q_0, \leq T)$, contains a neighborhood of q_0 for all neighborhoods V or q_0 and all times $T > 0$.
Mathematically, we may express this as:

$$\forall V \subseteq R^n, T > 0, \ \exists \, \varepsilon \ s.t. \ B_\varepsilon(q_0) \subseteq \mathcal{R}^V(q_0, \leq T) \tag{3.68}$$

Let's try to break down what this definition means. Looking at the conditions in the definition, we see that for a system to be small time locally controllable, we require that for *all* possible times T and sets of initial conditions V, we need the set of states reachable by the system from an initial state $q_0 \in V$ to contain a ball around q_0.



Above: the states reachable from q_0 form a ball around q_0.

Thus, if a system is small time locally controllable, we can drive it to a set of all nearby points in a small, finite amount of time! This is a highly desirable property in robotics, as we want to know if we're able to place our system where we want while respecting the constraints on its motion.
Now that we have this definition of controllability, we must determine how to actually *prove* that a system is small time locally controllable. We can use our results from the previous section, where we used the idea of a Lie algebra to express the possible motions along a set of vector fields. This idea brings us to the following famous theorem.[1]

Theorem 6 ***Chow's Theorem (Small Time Local Controllability)***
A system $\dot{q} = g_1(q)u_1 + ... + g_m(q)u_m$, where $u_i \in \mathbb{R}$ and $q \in \mathbb{R}^n$, is small time locally controllable at a point q if the Lie algebra of the vector fields:

$$\mathcal{L}(g_1, ..., g_m) = span\{g_1, ..., g_m, [g_1, g_2], ..., [g_{m-1}, g_m], [g_1, [g_1, g_2]], ...\} \tag{3.69}$$

[1] Note that the condition for Chow's theorem is stated more rigorously as the involutive closure of the distribution generated by $g_1, ..., g_m$ is equal to the tangent space to $\mathbb{R}^n$ at q.

Has dimension n at point q.

Thus, we conclude that for a system $\dot{q} = g_1(q)u_1 + ... + g_m(q)u_m$ to be able to reach any nearby state in finite time, we must be able to get n linearly independent vector fields from $g_1, ..., g_m$ and their Lie brackets.
Let's complete a quick example of determining the controllability of a system using Chow's theorem,. We'll now show that Raibert's hopper is a small time locally controllable system. Recall that earlier, we showed that the following is an equivalent control system for the hopping robot as it flies through the air:

$$\dot{q} = \begin{bmatrix} 1 \\ 0 \\ -\frac{m(l+d)^2}{I+m(l+d)^2} \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} u_1 = g_1(q)u_1 + g_2(q)u_2 \tag{3.70}$$

Where $q = [\psi, l, \theta]^T$ is the vector of generalized coordinates needed to describe the motion of the system. To show that the system is small time locally controllable, we must show that its Lie algebra, $\mathcal{L}(g_1, g_2)$, has the same dimension as the state vector, $n = 3$.
We can do this by showing we can find three linearly independent vector fields from the vector fields g_1, g_2, and their Lie brackets. Looking at g_1, g_2, we notice that they are already linearly independent! Thus, all we need is one more vector field to complete the process of showing the Lie algebra is three dimensional.
Let's try the following simple, first order Lie bracket as a candidate for our third linearly independent vector field:

$$[g_1, g_2] = \frac{\partial g_2}{\partial q} g_1(q) - \frac{\partial g_1}{\partial q} g_2(q) \tag{3.71}$$

First, we calculate the matrices of partial derivatives. Since g_2 is a constant vector field that doesn't depend on q, we know that:

$$\frac{\partial g_2}{\partial q} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.72}$$

What about $\frac{\partial g_1}{\partial q}$? Calculating each partial derivative and simplifying, we get:

$$\frac{\partial g_1}{\partial q} = \begin{bmatrix} \frac{\partial g_{11}}{\partial q_1} & \frac{\partial g_{11}}{\partial q_2} & \frac{\partial g_{11}}{\partial q_3} \\ \frac{\partial g_{12}}{\partial q_1} & \frac{\partial g_{12}}{\partial q_2} & \frac{\partial g_{12}}{\partial q_3} \\ \frac{\partial g_{13}}{\partial q_1} & \frac{\partial g_{13}}{\partial q_2} & \frac{\partial g_{13}}{\partial q_3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{-2m(l+d)I}{(I+m(l+d)^2)^2} & 0 \end{bmatrix} \tag{3.73}$$

Now, calculating out the expression of the Lie bracket in full, this leaves us with the following:

$$[g_1, g_2] = 0 - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{-2m(l+d)I}{(I+m(l+d)^2)^2} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{2m(l+d)I}{(I+m(l+d)^2)^2} \end{bmatrix} \tag{3.74}$$

Is the set $\{g_1, g_2, [g_1, g_2]\}$ linearly independent? If it is, we've found that the dimension of our Lie algebra (the span of g_1, g_2 and their Lie brackets) is equal to the dimension of the state vector, 3! If the set is not linearly independent, we can keep taking further, higher order Lie brackets.
To check if the set $\{g_1, g_2, [g_1, g_2]\}$ is linearly independent, we may place each vector field in the column of a matrix and check that the determinant of the matrix is nonzero.

$$\det \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{m(l+d)^2}{I+m(l+d)^2} & 0 & \frac{2m(l+d)I}{(I+m(l+d)^2)^2} \end{bmatrix} \stackrel{?}{=} 0 \tag{3.75}$$

Computing the determinant of this matrix using a symbolic calculator, we find:

$$\det \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{m(l+d)^2}{I+m(l+d)^2} & 0 & \frac{2m(l+d)I}{(I+m(l+d)^2)^2} \end{bmatrix} = \frac{2Im(l+d)}{(md^2+2mdl+ml^2+I)^2} \tag{3.76}$$

Thus, as long as we have that $l \neq -d$, the dimension of the Lie algebra of the system will be equal to 3, as we have three linearly independent vector fields generated from g_1, g_2, and their Lie bracket, $[g_1, g_2]$.
Since the dimension of the state vector is equal to three, we may therefore apply Chow's theorem to conclude that when $l \neq -d$, the system is small time locally controllable.

Chapter 4

Estimation

So far in our study of dynamical systems, control, and path planning, we've assumed that the state of our system is known with *absolute precision*. For instance, when designing a basic state feedback controller:

$$u = -Kx \tag{4.1}$$

We assume that we have full access to the state vector, x, of the system. Additionally, we assume that our knowledge of the state vector is perfect, and true to what the state of the system actually is. In the real world, however, these assumptions break down.

When interacting with robotic systems, we must gain information about the state of our systems through noisy *sensors*, which we can use to piece together the different components of our state vector as time passes. Since our sensors contain noise, they will *never* give us a perfect measurement of the true state of our system! Furthermore, we won't always be able to directly measure every component of the system's state vector.

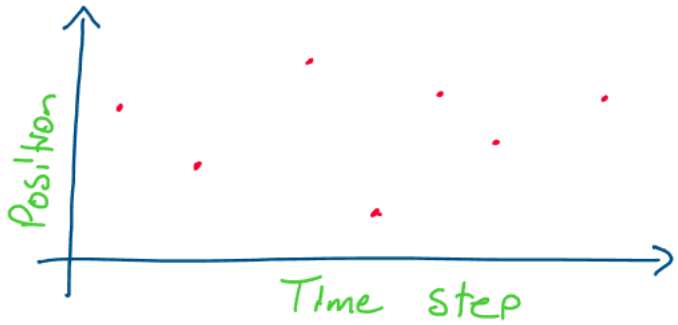
How can we get around these challenges? To solve the problem of filtering out noise from our measurements and reconstructing the entire state vector of our system from potentially incomplete measurements, we turn to the field of estimation.

In this section of the course, we'll study some fundamental concepts in the estimation of stochastic systems - systems with random noise. Before getting into the specifics of these techniques, let's develop some foundational material in probability and random processes.

4.1 Elements of Probability

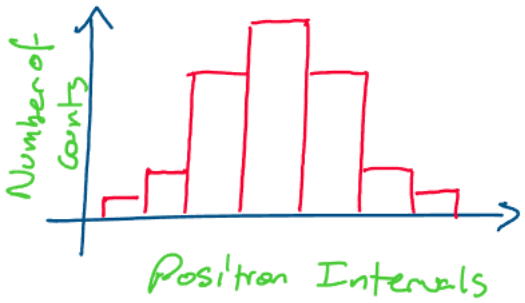
When developing strong estimates of the state of a system using noisy sensors, it's important that we have a strong understanding of how to model the noise the sensors actually experience! Let's try and understand the noise we'll be dealing with with a simple example.

Let's imagine that we're trying to measure the position of a turtlebot, which sits stationary on the ground, using an AR tag. In this case, even though the turtlebot stays still in physical space, the measurements of the turtlebot's position from the AR tag will *jump around* due to noise in the sensor's measurements.



Above: Due to noise, our measurements will jump around, despite the underlying true value remaining the same.

As we take more and more measurements of our turtlebot's state, we can keep track of the ranges our measurements fall into using a histogram. By counting the number of measurements that fall into certain distinct ranges, we may form a plot such as the following:



Above: We can keep track of the occurrences of ranges of measurements using a histogram.

As we can see from the image above, as we collect these measurements, we can begin to form a *distribution* of the measurements of the turtlebot - a relationship that describes the frequencies of certain measurements. Through applying the techniques of probability theory, we can use the different properties of this distribution of measurements to gain a prediction of where our turtlebot actually is in space!

Let's begin by reviewing some fundamental concepts in probability. As we move through this material, we'll gradually work our way back to understanding how noise interacts with robotic systems.

Our treatment of probability in this section will most closely follow Tomizuka's

notes on optimal control, *Optimization-Based Control* by Murray, and *Introduction to Probability* by Blitzstein and Hwang.

4.1.1 Probability Spaces

Let's begin our exploration of probability by setting up a simple problem. Suppose we have a single, six-sided die. Imagine that we want to find how likely it is that when we roll the die, we get a six. What tools do we need to treat this problem in the language of formal mathematics? Let's begin by making a few definitions.

First, we define an **experiment**. An experiment is something in which we observe the outcome of a random process. For instance, rolling a die and observing the number is considered to be an experiment in the language of probability, since the act of rolling a die has some inherent randomness.

Next, we define the **sample space**, which we denote with the letter Ω. Ω is the set of *all* possible outcomes in our probabilistic experiment. In the case of a rolling a single die, since the dice may land on the numbers 1 through 6, the sample space Ω would be the set:

$$\Omega = \{1, 2, 3, 4, 5, 6\} \tag{4.2}$$

As this set contains all of the possible outcomes of rolling a single die. When referring to individual outcomes contained within the sample space Ω, we use the letter ω. Using the conventions of set notation, we may indicate that ω is an *element* of the sample space Ω by writing:

$$\omega \in \Omega \tag{4.3}$$

When discussing the chances of certain outcomes or combinations of outcomes occurring, we use something called an **event**. Formally, every event, denoted by the letter S, is a subset of the sample space, Ω.

$$S \subseteq \Omega \tag{4.4}$$

Every possible event must be one of or a collection of some of the possible outcomes for our experiment. Typically, each event S will contain a set of outcomes ω that satisfy a certain condition that we're looking for. For example, in the example of rolling a dice S could be the set of outcomes where the dice lands on a 6.

How can we reason about the chance that a certain event will occur? The probability function, $\mathbb{P}(S)$, takes in an event S and returns a number between 0 and 1, called a **probability**, that tells us how likely that event is to occur. If the probability of an event $\mathbb{P}(S)$ is equal to 1, then the event S is *guaranteed* to occur, whereas if $\mathbb{P}(S) = 0$, the event S has *no chance* of occurring.

The probability function must satisfy the following three fundamental rules, known as the probability axioms:

Definition 36 ***Probability Axioms***
If Ω is a sample space the probability function $\mathbb{P}$ must satisfy the following properties for all events $S_1, S_2 \subseteq \Omega$:

1. ***Probability is always positive:*** $\mathbb{P}(S_1) \geq 0$
2. ***Probability of the outcome space is 1:*** $\mathbb{P}(\Omega) = 1$
3. ***Probability of disjoint events sum:*** $\mathbb{P}(S_1 \cup S_2) = \mathbb{P}(S_1) + \mathbb{P}(S_2)$ *if* $S_1 \cap S_2 = \emptyset$

The first condition states that probabilities must *always* be positive! Since an event has zero probability if it doesn't happen, having a negative probability doesn't have any meaning. The second condition states that the probability of *all* possible outcomes combined must be 1 - at least one outcome *must* happen when performing an experiment.
Finally, the last rule states that the probabilities of *disjoint* events sum. If two events, S_1 and S_2, have *nothing* in common, we can find the probability that S_1 *or* S_2 happens by adding the probabilities that they each happen individually. For instance, if S_1 is the event that our dice lands on a 3, and S_2 is the event that our dice lands on a 6, S_1, S_2 have nothing in common, as they represent totally different outcomes. We may write this as:

$$S_1 \cap S_2 = \emptyset \tag{4.5}$$

Where $S_1 \cap S_2$ is the intersection of S_1 and S_2, the set that contains all outcomes shared by S_1 and S_2, and $\emptyset$ is the **empty set**, the set that contains no elements. Thus, if we wanted to find the probability that we *either* roll a 3 or a 6, we could add the probabilities of each individual event:

$$\mathbb{P}(S_1 \cup S_2) = \mathbb{P}(S_1) + \mathbb{P}(S_2) = \frac{1}{6} + \frac{1}{6} = \frac{1}{3} \tag{4.6}$$

Since the events S_1 and S_2 of getting a three and getting a six are disjoint.
Note that in more advanced treatments of probability, the probability function $\mathbb{P}$ is said to be a probability *measure*, as $\mathbb{P}$ measures the "size" of the event S with respect to the "size" of the outcome space to find the probability of the event occurring.[1]
Oftentimes, instead of directly finding the probability that an event happens, it can be easier to solve for the probability that the event *doesn't happen*. We can formalize this idea mathematically with the idea of a **complement**.

[1] If you're interested in learning more about this, look into the field of *measure theory*, which forms the foundations for a rigorous construction of probability.

Definition 37 ***Complement of an Event***
The complement of an event $S \subseteq \Omega$, denoted S^c, is the set of all outcomes in Ω that do not belong to S.

Intuitively, if S is an event containing some outcomes, S^c is the event containing all of the other possible outcomes in the sample space. Using the probability axioms and the definition of a complement, we may develop the following useful result regarding the probability of events and their complements:

Proposition 6 ***Probability of a Complement***
If $S \subseteq \Omega$ is an event with a complement $S^c \subseteq \Omega$, the probability of S is computed:

$$\mathbb{P}(S) = 1 - \mathbb{P}(S^c) \tag{4.7}$$

Proof: The key to this result is that the intersection of S and S^c, $S \cap S^c$ is *empty*, since S^c contains everything in Ω that is not in S. Additionally, we know that the combination of S and S^c, $S \cup S^c = \Omega$, by the definition of the complement. Applying the probability axioms, this tells us:

$$\mathbb{P}(S \cup S^c) = \mathbb{P}(S) + \mathbb{P}(S^c) \tag{4.8}$$
$$\mathbb{P}(\Omega) = \mathbb{P}(S) + \mathbb{P}(S^c) \tag{4.9}$$
$$1 = \mathbb{P}(S) + \mathbb{P}(S^c) \tag{4.10}$$
$$\mathbb{P}(S) = 1 - \mathbb{P}(S^c) \tag{4.11}$$

This completes the proof! □

4.1.2 Random Variables and Vectors

Now that we've formed some basic definitions in probability, let's discuss some more interesting results! So far in our treatment of probability, we've discussed some basic properties of the probability function $\mathbb{P}$, but haven't yet considered what the function $\mathbb{P}$ might actually be.

Random Variables

We can begin answering this question in a more mathematical sense by defining something known as a **random variable**. Thus far, you may have noticed that our events, such as "the dice lands on a six," or "the dice lands on a six or on a three," are objects that simply exist in *words* - they're not yet purely mathematical expressions! To *abstract away* the language and physical descriptions associated with events, we introduce the following definition.

Definition 38 ***Random Variable***
Suppose we have a sample space Ω which contains events $S \subseteq \Omega$. A random variable is a function of events:

$$X(S) : S \to \mathbb{R} \tag{4.12}$$

That maps each event $S \subseteq \Omega$ to a real number $X(S) \in \mathbb{R}$.

Thus, a random variable helps us think about events in terms of numerical values. Let's make this definition a little bit more concrete by turning back to our example of rolling a single six sided die. In this case, we could use a random variable X to represent the number that turns up when we roll the die! We could then express the event "we roll a 6" through the mathematical expression:

$$X = 6 \tag{4.13}$$

This enables us to express the probability of rolling a six as:

$$\mathbb{P}(X = 6) = \frac{1}{6} \tag{4.14}$$

Where random variables become highly useful is in expressing the probability of more complex events! For instance, if we wanted to find the probability that we rolled something between a 2 and a 4, we could write:

$$\mathbb{P}(2 \leq X \leq 4) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2} \tag{4.15}$$

Random variables and their associated notation will be immensely convenient moving forwards as we generalize our study of probability to arbitrary events. As you can see from the example above, by assigning real values numbers to arbitrary events, we can see events and probability itself in a more precise, mathematical light.

Our random variable X from above is an example of a **discrete random variable**. A discrete random variable is a random variable that can only take on a *countable number of values*![2] For instance, the random variable X representing our die roll can only have the values of $1, 2, 3, 4, 5, 6$.

In robotics, where our random variables might refer to things such as positions in space, having just these discrete random variables *won't* be good enough! After all, even in one dimension, there are an uncountably infinite number of positions we can travel to. If a random variable can take on any value in the real numbers, is is said to be a **continuous random variable**. This implies that the number of values a continuous random variable can take on are *infinite*.

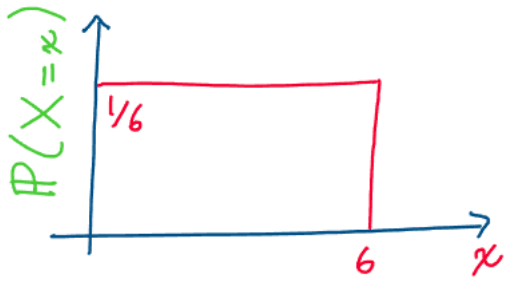
[2] Countable doesn't necessarily mean finite! The set of integers is a countable set of numbers, for instance - you can think of the integers as a typical set of countable numbers. The real numbers, on the other hand, are *uncountable*.

Probability Density and Cumulative Distribution Functions

Now that we've defined random variables, we may think about how the probability function $\mathbb{P}$ is actually constructed for different experiments.
Once again, we'll begin our discussion by thinking about the probabilities associated with the different outcomes of a die roll. Let's come up with a function $\mathbb{P}(X = x)$ that tells us the probability that we will roll x on any given trial. Since the die is equally likely to land on any of its sides, we know that for all integer values of x between 1 and 6:

$$\mathbb{P}(X = x) = \frac{1}{6} \tag{4.16}$$

This function holds for all integer values of x, as landing on any one of the six sides of a die is equally likely as landing on another. We can plot this function on the domain $[1, 6]$ as follows:



Above: A dice has equal probability of landing on any one number.

Notice how the total area under this curve is equal to 1, the probability of the total outcome space Ω occurring.
What other probabilities might we be interested in? In addition to knowing the probability that the dice will land on any one value, perhaps it would be useful to know the probability that the dice lands on something *less than or equal to* a certain value x. For the die, we can express $\mathbb{P}(X \leq x)$ on the same domain as above as follows:

$$\mathbb{P}(X \leq x) = \frac{x}{6} \tag{4.17}$$

Where x is an integer between 1 and 6.
We can think of this probability as something that *accumulates* the probabilities of all of the ways X can be less than or equal to a certain value x.
Let's discuss the equivalents of these two functions for a continuous random variable X! Recall that one of the conditions for a random variable to be continuous is that it must have an infinite number of possible values! With this in mind, we define the following function.

Definition 39 ***Cumulative Distribution Function (CDF)***
The cumulative distribution function (CDF) of a continuous random variable X, denoted $F(x)$, is defined to be the probability that $X \leq x$.

$$F(x) = \mathbb{P}(X \leq x) \tag{4.18}$$

Just like with the (discrete) case of the dice, a cumulative distribution function *accumulates* the probabilities of all of the ways a random variable can have a value less than or equal to a value x. Since a continuous random variable X has values across *all* of the real numbers, we may equivalently write the cumulative distribution function of X as:

$$F(x) = \mathbb{P}(-\infty < X \leq x) \tag{4.19}$$

Let's now think about the other function we defined for the dice, which gave the value that $\mathbb{P}(X = x)$. For the case of a continuous random variable, the equivalent of this function is much more subtle. Since X now has an *infinite* range of possible values, the probability that X will take on *any specific* value of x will now be zero! How can we define a meaningful function that tells us something about each individual value of a random variable?
Consider the following definition:

Definition 40 ***Probability Density Function (PDF)***
The probability density function (PDF) of a random variable X, is defined to be the function $f(x)$ such that:

$$\mathbb{P}(a \leq X \leq b) = \int_a^b f(x)dx \tag{4.20}$$

For a differentiable cumulative density function $F(x)$, the probability density function may therefore be found by differentiating $F(x)$ with respect to x:

$$f(x) = \frac{dF}{dx} \tag{4.21}$$

By *integrating* the probability density function over a range of values, we may find the probability that our random variable X falls within a certain range! It's *extremely important* to note that the value of the PDF, $f(x)$, does not give the probability that $X = x$. The probability that $X = x$ is always equal to zero for a continuous random variable X.
This definition tells us that we may derive the cumulative distribution $F(x)$

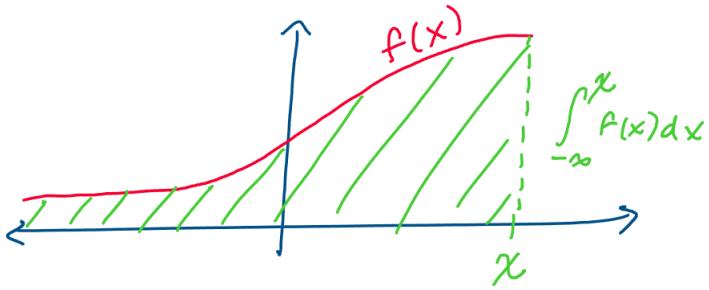
from a density $f(x)$ of a random variable X by computing the integral:

$$F(x) = \int_{-\infty}^{x} f(\chi) d\chi \tag{4.22}$$

Furthermore, by the fundamental theorem of calculus, we may calculate the probability that X lies between a and b, where $a \leq b$, through computing:

$$\mathbb{P}(a \leq X \leq b) = F(b) - F(a) \tag{4.23}$$

These useful facts will help us switch between the CDF and PDF of a continuous random variable, and easily calculate the probabilities associated with ranges of values.



Above: We can integrate the density function to obtain a probability.

Since both the PDF and CDF tell us exactly how to find the probability that our random variable X will fall inside a certain range of values, we may specify either the PDF or CDF to entirely characterize a random variable!
Using the axioms of the probability function $\mathbb{P}$, we may arrive at the following rules for the probability density function of any random variable.

Proposition 7 ***Density Properties***
If X is a continuous random variable with density $f(x)$, $f(x)$ must have the following properties.

1. ***Non-Negative:*** *$f(x) \geq 0$ for all $x \in \mathbb{R}$.*
2. ***Integrates to 1:*** *The integral of the density over the entire real line must be equal to 1.*

$$\int_{-\infty}^{\infty} f(x) dx = 1 \tag{4.24}$$

The first property comes from the fact that probability must always be positive.

To avoid the possibility of having a negative probability in a region, we constrain the PDF to always be greater than or equal to zero. The second property comes from the fact that $\mathbb{P}(\Omega) = 1$, as it's a certainty that our random variable X will obtain *some* value in the real numbers.
Notice that the fact that the probability density is always ≥ 0 tells us a useful fact about the cumulative distribution of a random variable: the CDF of a random variable x is either constant or increasing at any value of x, as the CDF may be found by taking the integral of the PDF. Further, from the second property of the probability density function, we know that the CDF $F(x)$ must have a value between 0 and 1 for all x.
When working with multiple random variables, it's often important to think about the relationship between random variables. For instance, the outcome of a random variable:

$$X = \{\text{It will rain today}\} \tag{4.25}$$

Might have an impact on the outcome of the random variable:

$$Y = \{\text{I will use an umbrella today}\} \tag{4.26}$$

We can express the idea of two random variables being "related" with the idea of a joint distribution.

Definition 41 ***Joint Cumulative Distribution Function***
Given two random variables X, Y the joint cumulative distribution function $F(x, y)$ of the two random variables expresses the probability that $X \leq x$ and $Y \leq y$.

$$F(x, y) = \mathbb{P}(X \leq x, Y \leq y) \tag{4.27}$$

Note that the comma in the joint cumulative distribution definition, $F(x, y) = \mathbb{P}(X \leq x, Y \leq y)$, simply means "and." We'll often use "joint CDF" as shorthand for the name of this function.
Just like we can find a single random variable's probability density function by differentiating its cumulative density function, we may do something similar to find a *joint density* of multiple random variables!

Definition 42 ***Joint Density Function***
The joint density function $f(x, y)$ of two random variables X, Y with a joint CDF $F(x, y)$ is computed:

$$f(x, y) = \frac{\partial^2}{\partial x \partial y}(F(x, y)) \tag{4.28}$$

Note that using the rules of partial derivatives, it doesn't matter if we take the partial derivative with respect to x and then y or y and then x - both will produce equivalent expressions.
Using the joint density function $f(x, y)$, we may calculate the probability that the random variables X and Y will fall into certain ranges of values by taking a *double* integral. For instance, we can compute:

$$\mathbb{P}(x_1 \leq X \leq x_2, y_1 \leq Y \leq y_2) = \int_{x_1}^{x_2} \int_{y_1}^{y_2} f(x, y) dy dx \tag{4.29}$$

Using the concept of a joint distribution, is there some way we can mathematically verify if a relationship between two random variables actually exists? We may define what it means for two random variables to be independent as follows:

Definition 43 ***Independence of Random Variables***
Let X, Y be two random variables. If for all values of x, y, is is true that:

$$\mathbb{P}(X \leq x, Y \leq y) = \mathbb{P}(X \leq x)\mathbb{P}(Y \leq y) \tag{4.30}$$

Then X and Y are said to be independent random variables.

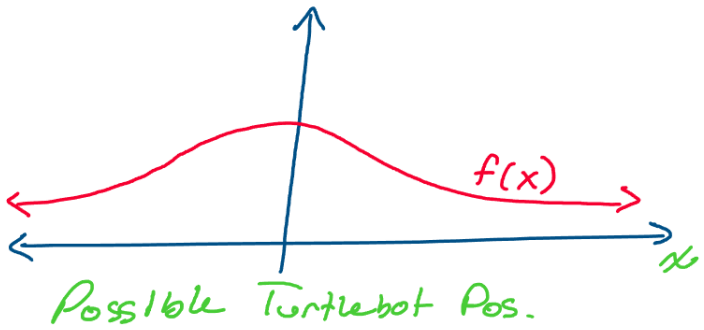
If two random variables are independent, then the outcome of one random variable will have *no effect* on the outcome of another random variable! For instance, since the value I get on one roll of a die has no influence on the values I get on future rolls, we consider each die roll to be independent.
The definition of independence also allows us to make a conclusion about the relationship between the joint density of two independent random variables and the individual densities of the random variables. If X, Y are independent random variables with densities $f_x(x), f_y(y)$ and a joint density $f(x, y)$, then the joint density is actually the product of the two individual densities:

$$f(x, y) = f_x(x) f_y(y) \tag{4.31}$$

Note that in literature, the individual densities of the random variables, $f_x(x)$ and $f_y(y)$, are referred to as the **marginal densities** of X and Y.

Expectation and Variance

Let's briefly think back to our example of measuring the position of a turtlebot in space. As time passes, the measurements we take of the turtlebot's position jump around due to noise and form a *distribution* describing the possible states of the turtlebot!



Above: We can model the possible measured states of our turtlebot using a distribution.

How can we use this distribution to find the *most likely*, or *expected*, position of the turtlebot? Let's simplify this problem somewhat for the sake of discussion. Suppose that the x position of the turtlebot can be any one of the k values $x_1, ..., x_k$, and that we represent the x position of the turtlebot using the discrete random variable X. Intuitively, we can compute the most likely position of the turtlebot by averaging all of our measurements!
If each position x_i appears n_i times when taking measurements, we can compute the most likely value of this random variable by computing the weighted sum:

$$x_{expected} = \frac{n_1 x_1 + ... + n_k x_k}{n_1 + ... + n_k} = \frac{n_1}{N} x_1 + ... + \frac{n_k}{N} x_k \tag{4.32}$$

Where $N = n_1 + ... + n_k$ is the total number of measurements we have taken. After we take *many* measurements, in the limit as $N \to \infty$, we may show that each ratio, $\frac{n_i}{N}$, will actually converge to the probability that each x_i is measured by the sensor! Thus, in the limit, we have:

$$x_{expected} = \mathbb{P}(X = x_1)x_1 + ... + \mathbb{P}(X = x_n)x_n \tag{4.33}$$

What we have above is the probability of each outcome multiplied by the value of that outcome! Thus, for the case of a discrete random variable, the average, or "expected value," of a random variable is a weighted sum of the different possible values of the random variable.
Can we generalize this idea of finding the most likely value of a simple random variable to a continuous random variable with an arbitrary density function $f(x)$? The **expected value** of a random variable, defined to be the average value of the random variable across its probability distribution, accomplishes this generalization. To calculate the expected value of a continuous random variable, we perform a similar operation to the above, now using an integral in the place of a sum and density in the place of probability.

Definition 44 ***Expected Value***
The expected value of a random variable X with density $f(x)$ is defined:

$$\mu_X = \mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x) dx \tag{4.34}$$

Expected value is also called the mean or average of a random variable, and is a deterministic (non-random) quantity.

Notice how we perform an operation similar to that which we discussed above! To find the expected value, we use an integral to accumulate the products of random variable values and densities.
The expression $\mathbb{E}[X]$ is typically read as the "expected value" or "expectation" of X. Note that the variable μ_X is also commonly used as shorthand for the expected value of a random variable X. Expected value has the following important property we'll make extensive use of:

Proposition 8 ***Linearity of Expectation***
Expectation is a linear function of random variables. If X, Y are two jointly distributed random variables and $a, b \in \mathbb{R}$ are scalar constants, then:

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y] \tag{4.35}$$

Proof: Let's assume that X, Y are jointly distributed random variables with a joint probability distribution $f(x, y)$. Let's try expanding the expression $\mathbb{E}[aX + bY]$ in terms of the integral definition of expected value, and see what we find! We may calculate $\mathbb{E}[aX + bY]$ by performing a double integral over the joint distribution of X and Y.

$$\mathbb{E}[aX + bY] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (ax + by) f(x, y) dx dy \tag{4.36}$$

Note how in the above, we integrate over *both* x and y when taking the expected value, as we are using the joint distribution $f(x, y)$, rather than an individual marginal distribution of one of the random variables. Let's split up the x and y terms in this integral.

$$\mathbb{E}[aX + bY] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (ax + by) f(x, y) dx dy \tag{4.37}$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} ax f(x, y) dx dy + \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} by f(x, y) dx dy \tag{4.38}$$

Now, by changing the order of integration, we may extract the marginal distributions of each of the random variables! Performing this operation and pulling the constants out of the integrals, we find:

$$\mathbb{E}[aX + bY] = a\int_{-\infty}^{\infty} x \int_{-\infty}^{\infty} f(x,y)dydx + b\int_{-\infty}^{\infty} y \int_{-\infty}^{\infty} f(x,y)dxdy \tag{4.39}$$

Performing the inner integrals, we may prove that we get the marginal density functions of X and Y, $f_x(x)$ and $f_y(y)$:

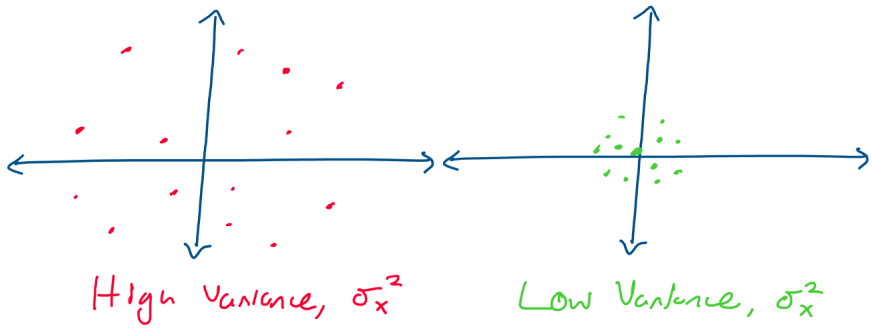
$$\mathbb{E}[aX + bY] = a\int_{-\infty}^{\infty} x f_x(x)dx + b\int_{-\infty}^{\infty} y f_y(y)dy \tag{4.40}$$

Miraculously, after performing this operation, we see that the integrals that remain provide the expected values of X and Y! Thus:

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y] \tag{4.41}$$

This completes the proof! □

Now that we've developed some basic properties of random variables, we can think about some other properties that might characterize the probabilistic nature of random variables.
Suppose, for instance, that we have two noisy sensors that allows us to determine the position of our robot. Imagine that after taking a few measurements of our robot's position with both of these sensors, we arrive at the following plots:



Above: A wide spread of data will have high variance, while a tight cluster will have low variance.

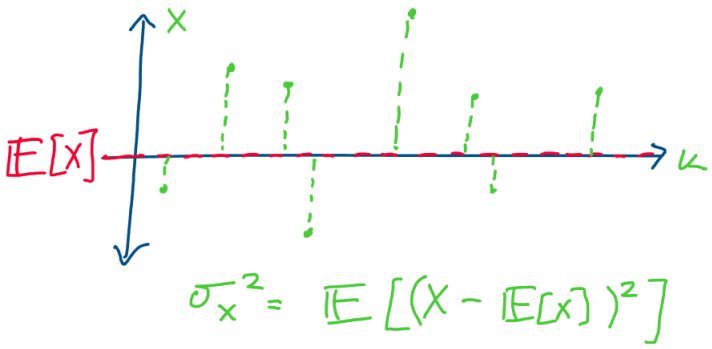
As we can see from the image above, even though both of the sensors might produce data with the same average (expected value), as time goes on, the *spread* of the measurements taken by the two sensors might differ significantly! To characterize how "spread out" the values of a random variable will be, we may use a quantity known as **variance**.

Definition 45 ***Variance***
Suppose X is a random variable with a probability density function $f(x)$. The variance of X, σ_X^2 is defined:

$$\sigma_X^2 = \mathbb{E}[(X - \mu_X)^2] = \int_{-\infty}^{\infty} (x - \mu_X)^2 f(x) dx \tag{4.42}$$

Where $\mu_X = \mathbb{E}[X]$, the expected value of X.

As we can see from the definition above, the variance of a random variable tells us the *expected* squared distance of the random variable to its mean! The higher the squared distance between the random variable and the mean, the wider the spread of the values of the random variable will be.
Note that we take the *squared distance* to the mean so symmetric spreads around the mean of the variable don't cancel one another out and give zero variance.



Above: By finding the expected square distance of our random variable to the mean, we can find variance.

Based on the sensor measurements discussed above, we would expect the first sensor to have a high variance in its measurements, as it has a wide spread of data, and the second sensor to have a small variance, as the measurements as its measurements are clustered tightly around the average.
Notice how in the symbol for variance, σ_X^2, we have an exponent! If we take the square root of variance, which will give us σ_X, we're left with a quantity known as the **standard deviation** of a random variable. Standard deviation may be used to help us understand the spread of the distribution in non-squared units, as opposed to the squared distance values that variance provides.
Closely related to the concept of variance is **covariance**, which shares a similar definition. Instead of just telling us about the spread of a single random variable, covariance helps us express the relationship between two random variables.

Definition 46 ***Covariance***
Suppose X and Y are jointly distributed random variables with expected values μ_x and μ_y. The covariance of X and Y, denoted Σ_{xy}, is defined:

$$\Sigma_{XY} = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] \tag{4.43}$$

Let's take a moment to think about what this definition expresses. Recall that the joint probability distribution between X and Y expresses a relationship between X and Y. Let's think about some different possibilities for this relationship and see where covariance comes into the picture.
First, imagine that we have the following relationship: if X frequently takes on a value higher than its mean, then Y will also frequently take on a value higher than its mean. This means that $X - \mu_X$ and $Y - \mu_Y$ will both frequently be positive, which will give a positive covariance Σ_{XY}.
Similarly, if X frequently being *lower* than its mean implies that Y will frequently be lower than its mean, $X - \mu_X$ and $Y - \mu_Y$ will both frequently be negative, and Σ_{XY} will again be positive.
Because of these relationships, we say that the covariance of X and Y, Σ_{XY}, measures the tendency of two random variables to move up or down in their values together. Using this idea, if X and Y have a covariance of 0, there will be no correlation between the changes in values of X and Y. Thus, if $\Sigma_{XY} = 0$, we say that X and Y are **uncorrelated** random variables. Note that all independent random variables are uncorrelated.
Notice how our notation for covariance means that the variance of a single random variable, σ_X^2, may also be denoted Σ_{XX}, as the definitions of Σ_{XX} and σ_X^2 will be the same.

$$\Sigma_{XX} = \mathbb{E}[(X - \mu_X)(X - \mu_X)] = \mathbb{E}[(X - \mu_X)^2] = \sigma_X^2 \tag{4.44}$$

Random Vectors

Thus far, we've constrained our discussion of probability to the theory of *scalar random variables*. In robotics, however, we know that when dealing with quantities such as position and velocity, we require *vectors*.
How can we extend the concepts of probability we've developed thus far to the vector case? By placing a set of jointly distributed random variables together in a vector, we can form **random vectors** that describe higher dimensional probabilistic systems.
Let's define random vectors a little bit more formally to assign some mathematical meaning to this idea.

Definition 47 ***Random Vector***
Let $X_1, X_2, ..., X_n$ be scalar random variables with a joint probability density function $f(x_1, ..., x_n)$. The vector:

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} \in \mathbb{R}^n \tag{4.45}$$

Is called a random vector.

As we can see from the definition above, random vectors are simply collections of potentially related random variables that have been organized in a vector.
Using this definition, we may extend concepts from scalar random variables up to random vectors without too much trouble. For instance, the expected value of a random vector $X = [X_1, ..., X_n]^T$ is defined:

$$\mathbb{E}[X] = \begin{bmatrix} \mathbb{E}[X_1] \\ \vdots \\ \mathbb{E}[X_n] \end{bmatrix} = \begin{bmatrix} \mu_{X_1} \\ \vdots \\ \mu_{X_n} \end{bmatrix} \tag{4.46}$$

As we can see from the above, the expected value of a random vector is simply a vector containing the expected values of each random variable within the random vector.
When defining quantities such as variance and covariance for random vectors, we must change our approach slightly! Recall that the variance of a single random variable X is defined:

$$\sigma_X^2 = \mathbb{E}[(X - \mu_X)^2] \tag{4.47}$$

If we were to try and directly apply this definition to a vector, however, we wouldn't be able to, as we can't take the exponent of a vector quantity! Similarly, we can't define the covariance of two random vectors X and Y using our original definition:

$$\Sigma_{XY} = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] \tag{4.48}$$

Since we can't directly take the product of two vectors. To get around these problems, we redefine the covariance in a more general sense as follows:

Definition 48 ***Covariance of Random Vectors***
Let $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}^m$ be two random vectors. The covariance of X and Y, Σ_{XY}, is defined to be the matrix:

$$\Sigma_{XY} = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)^T] \in \mathbb{R}^{n \times m} \tag{4.49}$$

Thus, for the more general case of a random vector, covariance provides us with a *matrix*, each entry of which expresses the relationship between the elements of the random vector X and the elements of the random vector Y.
Note that instead of using variance to describe the spread of a single random vector X, moving forward we'll simply use the covariance of the random vector X with itself:

$$\Sigma_{XX} = \mathbb{E}[(X - \mu_X)(X - \mu_X)^T] \in \mathbb{R}^{n \times n} \tag{4.50}$$

Note that in practice, the covariance of a vector with itself is often written as Σ_X for short.
Notice how these matrix definitions reduce to our original definitions for variance and covariance when $n = 1$! Just as with the $n = 1$ case, if the covariance of two random vectors X and Y is the zero matrix, we say that the random vectors are uncorrelated.

4.1.3 The Gaussian Distribution

Now that we've covered a fairly significant body of work in probability, we can discuss one of the most important distributions in all of probability, the **Gaussian (normal) distribution**. For a single scalar random variable $X \in \mathbb{R}$, the Gaussian distribution is defined according to the following probability density function:

Definition 49 ***Gaussian Distribution***
Let $X \in \mathbb{R}$ be a scalar random variable. X is said to be Gaussian (normally) distributed with mean μ_X and variance σ_X^2 if it has the probability density function:

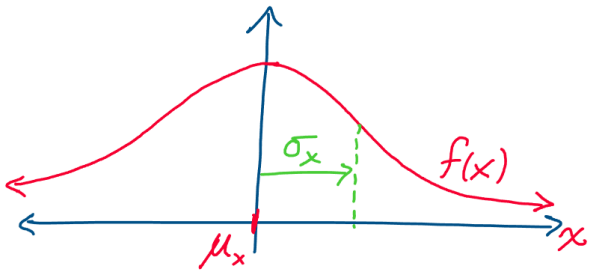
$$f(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left(-\frac{(x - \mu_X)^2}{2\sigma_X^2}\right) \tag{4.51}$$

Where exp *is the exponential function e^x.*

Note that if X is a random variable with a Gaussian probability density function of mean μ_X and variance σ_X^2, we often specify its distribution using the notation:

$$X \sim \mathcal{N}(\mu_X, \sigma_X^2) \tag{4.52}$$

Where $\sim$ means "is distributed as." Note that the symbol $\mathcal{N}$ comes from the alternate name of the Gaussian distribution: the *normal* distribution.
When plotted on a set of axes, the Gaussian density forms a smooth, symmetric bell-shaped curve that extends to $+\infty$ and $-\infty$.



Above: A Gaussian PDF of mean μ_X and variance σ_X^2.

Let's extend the definition of the Gaussian distribution from a single random variable X to a random vector $X = [X_1, ..., X_n]^T \in \mathbb{R}^n$.

Definition 50 ***Multivariate Gaussian***
If X is Gaussian distributed random vector with a mean $\mu_X \in \mathbb{R}^n$ and a covariance matrix $\Sigma_X \in \mathbb{R}^{n \times n}$, we may define its probability density function according to the multivariate Gaussian density function:

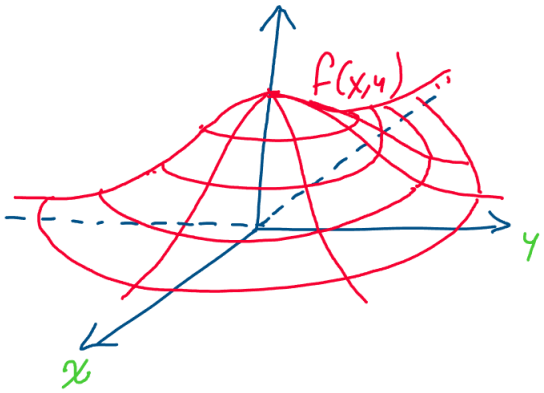
$$f(x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_X|}} \exp\Big(-\frac{1}{2}(x - \mu_X)^T \Sigma_X^{-1} (x - \mu_X) \Big) \tag{4.53}$$

Where $|\Sigma_X|$ is the determinant of the covariance matrix Σ_X.

In this case, we may indicate that X is Gaussian distributed using the same shorthand as before:

$$X \sim \mathcal{N}(\mu_X, \Sigma_X) \tag{4.54}$$

Similarly to the one dimensional Gaussian PDF, the multivariate Gaussian PDF takes on a symmetric bell-shape that extends to infinity in all directions. If $X \in \mathbb{R}^2$ is a Gaussian random vector, for instance, we may plot its PDF as:



Above: A multivariate Gaussian density.

The Gaussian distribution is of fundamental importance in probability theory. Amazingly, it can be proven for many distributions that by adding independent random variables of the same distribution, we get a random variable whose distribution approaches a Gaussian as the number of variables in the sum approach infinity!
This famous result, known as the **central limit theorem**, hints at the universal importance of the Gaussian distribution across probability theory.
Because of this property and many others, the Gaussian distribution will be at the core of our developments in robotic state estimation. Note that in our treatment of this material, we'll avoid going too deep into the mathematical properties of this distribution, and will rather keep our focus on the major highlights and results.
For a more mathematically detailed treatment of this material, you're encouraged to reference chapter 2 of *Optimal Control and Estimation* by Stengel.

4.1.4 Conditional Probability

Let's take a moment to bring our discussion of probability back to robotics. When taking measurements of robotic systems with noisy sensors, we're often faced with the problem of finding the best estimate of the current state of the system based on measurements we've taken in the past.
Is there some way we can use probability to best inform our estimate of the current state of the system using results we've observed in the past? The concept of *conditional probability* will help us answer this question, and will allow us use information we've observed about our system to make more accurate predictions.
Let's discuss some of the core concepts of conditional probability!
Imagine we have two random variables, X and Y, which have a joint probability density function $f(x, y)$. We know that the joint distribution $f(x, y)$ describes some sort of relationship between the random variables X and Y.
Using this joint distribution, can we make use of the relationship between X and Y and find how our knowledge of X changes when we know Y has a certain value? We may define the following probability density function, known as the **conditional density**, to answer this question.

Definition 51 ***Conditional Density Function***
Suppose X and Y are jointly distributed random variables or vectors with joint density $f(x, y)$. If we know that Y has the value y_1, we define the conditional density of X given $Y = y_1$ to be the function:

$$f_{X|Y=y_1}(x) = \frac{f(x, y_1)}{f_Y(y_1)} \tag{4.55}$$

Where f_Y is the marginal density of Y.

Based on the definition above, we may find the *conditional* distribution of X given $Y = y_1$ by considering the joint distribution of X and Y, and normalizing by the value of the density of Y when $Y = y_1$. This gives us a new distribution for X based on the relationship between X and Y and the knowledge that $Y = y_1$. Note that the notation $X|Y = y_1$, which is used in conditional probabilities, is read as "X given $Y = y_1$."
Using the conditional distribution, we define **conditional expectation**, which gives the expected value of X given $Y = y_1$, as:

$$\mu_{X|Y=y_1} = \mathbb{E}[X|Y = y_1] = \int_{-\infty}^{\infty} x f_{X|Y=y_1}(x)dx \tag{4.56}$$

Observe that the only difference between the definition of conditional expectation and standard expectation is that we use the *conditional density* of X given $Y = y_1$ instead of the standard unconditional density of X when taking the expected value integral.
Using conditional expectation, we define the conditional covariance of a random variable or vector X as:

$$\Sigma_{X|Y=y_1} = \mathbb{E}[(X - \mu_{X|Y=y_1})(X - \mu_{X|Y=y_1})^T] \tag{4.57}$$

Where the only difference to the standard definition of covariance is that we use the *conditional expectation* of X rather than the standard unconditional expectation in the definition of covariance.
By examining these quantities, we can determine how knowing $Y = y_1$ influences our knowledge of the properties of X. Note that if X and Y are independent random variables or vectors, knowing the value of Y should give us *no new information* about the values of X!
Because of this, in the case where X and Y are independent, the conditional density of X given $Y = y_1$ is simply the same as the original marginal density of X, as knowing the value of Y gives us no new information about X.

$$f_{X|Y=y_1}(x) = f_X(x) \text{ (For } X, Y \text{ independent R.V.s)} \tag{4.58}$$

4.1.5 Random Processes

In robotics, we know that we can use discrete time sequences to describe the behavior of robotic systems. Recall that earlier, we developed models for our systems of the form:

$$x_{k+1} = Ax_k + Bu_k \tag{4.59}$$

Where k is an integer that describes the current *time step* of the system. Note that the notation x_k is equivalent to $x(k)$ - the two can be used interchangeably depending on which is more convenient.
Using such relationships, we were able to describe the motion of discrete time systems through sequences of the form:

$$x_0, x_1, x_2, ... \tag{4.60}$$

A key property of these sequences was that they were *deterministic* - there was no randomness involved in the description of these sequences at each point in time, and we therefore assumed perfect knowledge of x_k.
Let's think about how we can apply the tools of probability to describe these sequences when each term in the sequence is not a deterministic vector x_k, but rather a *random vector* X_k.

$$X_0, X_1, X_2, ... \tag{4.61}$$

After developing some theory for random sequences, we'll be ready to bring our knowledge of probability back to robotic systems to solve practical problems in state estimation!
Fortunately, the probabilistic tools we've developed thus far extend nicely to the description of random sequences. One difference to note is that the probability distribution of our random vector X_k now has the potential to change with k. At each time step k, we may therefore define the expected value of X_k using our standard definition as:

$$\mu_{X_k} = \mathbb{E}[X_k] = \int_0^\infty x f(x,k) dx \tag{4.62}$$

Where the density of our random vector X_k now has the potential to change with respect to k. Note that in some literature, when it is clear that the random vector in question is X, the value μ_{X_k} is simply written in shorthand as μ_k.
When defining the covariance of a random vector sequence, we have to be a little bit more precise! Since the properties of the density of X now have the potential to change with k, it'll be informative not only to take the covariance of X_k with X_k, but also to take the covariance of X_k with X_j - the random vector at a different point in the sequence! This brings us to the following definition:

Definition 52 ***Auto-Covariance***
The auto-covariance of a random vector sequence X_k is defined to be the covariance of X between two points, k and j, in time:

$$\Sigma_{X_k X_j} = \mathbb{E}[(X_k - \mu_{X_k})(X_j - \mu_{X_j})^T] \tag{4.63}$$

When it is clear that the random vector in question is X, $\Sigma_{X_k X_j}$ may be written in shorthand as Σ_{kj}.
This definition helps us express the relationship between the random vector sequence X_k and itself at different points in time! With this idea in mind, let's consider the following scenario. Suppose the auto-covariance of a random vector sequence X_k has the following property, where $\Sigma_X \neq 0$:

$$\Sigma_{X_k X_j} = \begin{cases} \Sigma_X & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases} \tag{4.64}$$

In this case, we see that X_k is *only* correlated with itself, and no other X_j in the random sequence! In this case, the next random vector in the sequence has *no relation* to the current random vector in the sequence! This property frequently appears when describing the noise experienced by sensors.
Just like we can define a covariance between a random sequence and itself at different points in time, we can define the a covariance between two entirely different random sequences.

Definition 53 ***Cross-Covariance***
Suppose $X_k \in \mathbb{R}^n$ and $Y_k \in \mathbb{R}^m$ are two random sequences. The cross-covariance between X_k at time k and Y_k at time j is defined:

$$\Sigma_{X_k Y_j} = \mathbb{E}[(X_k - \mu_{X_k})(Y_j - \mu_{Y_j})^T] \tag{4.65}$$

This quantity expresses the relationship between the elements of X at time k and Y at time j. If $\Sigma_{X_k Y_j} = 0$ for all values of k and j, the sequences X_k and Y_k are said to be **uncorrelated**.
Using these definitions, we may easily extend the ideas of conditional expectation and covariance from single random vectors to sequences of random vectors by replacing densities with conditional densities and expectations with conditional expectations. Let's get to work on applying these concepts to robotic systems!

4.2 Stochastic Estimation

Let's tackle the problem of *optimal state estimation* in robotics. We know that the sensors which provide us with position, orientation, and velocity information about our system *all* experience noise. How can we filter this noise out to provide the best possible estimate of where our system actually is?
In this section, we'll piece together the Kalman filter, a famous filtering technique with extensive applications in robotics and control. Let's begin our development of this technique by formally analyzing the effect of noise on robotic systems.

4.2.1 Stochastic Dynamical Systems

In the previous section, we performed an expository discussion of *random vector sequences*, mathematical objects which model processes that evolve randomly in time. When modeling the noise experienced by systems, these random vector sequences will be an incredibly useful tool. Let's see how this is!
We'll begin by reviewing the formulation for a discrete time linear system, which will be the cornerstone of our results in this section. We stated that in discrete time, we may model a linear system using the following set of equations:

$$x_{k+1} = Ax_k + Bu_k \tag{4.66}$$
$$y_k = Cx_k \tag{4.67}$$

Where $x_k \in \mathbb{R}^n$ indicates the state vector of the system at time k, $u_k \in \mathbb{R}^m$ represents the input to the system at time k, and $y_k \in \mathbb{R}^p$ is the output of the system at time k. In the past, when discussing the control of dynamical systems, we thought of the output y_k of the system to be a vector containing the variables we want to control.
In the context of *estimation*, however, we define the output of the system to be a vector of variables that we can *measure*. For instance, if we had a turtlebot with a state vector:

$$x_{state} = \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} \tag{4.68}$$

But our sensors only gave us the ability to directly measure x and y, the output of our system would be the vector:

$$y_{out} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.69}$$

As these are the two quantities we have the ability to measure.
Let's think about how we can effectively model random noise for this class of systems. When discussing the effect of noise on dynamical systems such as the above, we will consider two separate classes: **process noise** and **measurement noise**. Let's discuss the differences between these two classes.

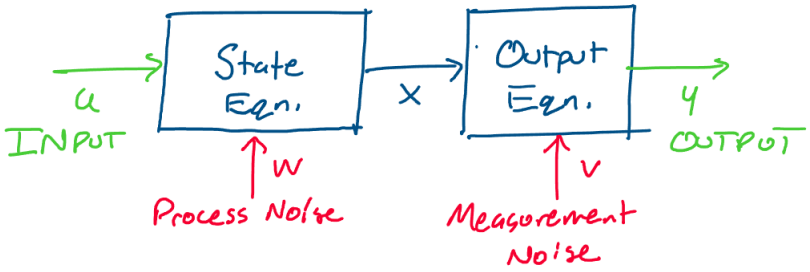
The process noise is a vector of noise that impacts the actual evolution of the system's state vector. For instance, if a random disturbance were to be applied to the turtlebot at each step in time, the disturbance would be considered an example of process noise! If we fly a quadrotor through an area with winds that change randomly with time, the force the winds apply to the quadrotor would also be considered process noise.
Since this noise *directly* impacts how the state vector of the system changes, the total process noise is a term that appears directly in the state equation, the equation which governs the dynamics of the system.
We can model the process noise by adding a random vector sequence $w_k \in \mathbb{R}^n$ to the state equation! To model the effect of process noise disturbances on the linear system, we therefore use a state equation of the form:

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{4.70}$$

Since the equation determining the change of the state vector in time now has a random term w_k in it, we know that the overall evolution of the state vector x_k must be random! Thus, x_k itself becomes a random vector sequence when we add process noise into the system. When a system becomes noisy and is governed by random processes, we say it is **stochastic**.
Let's discuss the second class of noise. Whereas process noise directly influences how the actual state vector of the system changes, measurement noise simply influences how noisy our sensor measurements of the system are - it has no actual influence on the dynamics of the system.
If we were to have zero sensor noise, we would be able to *perfectly* read the outputs of the system as time progresses. The more measurement noise we have, the harder it is to get an accurate reading of the output of the system.
Since the measurement noise *only* influences the measurements we take of the system, it will only appear in the output equation of the system. We can model measurement noise through a random vector sequence $v_k \in \mathbb{R}^p$, where p is the size of the output vector. By adding this noise to the output equation, we can accurately model real-world noisy sensors.

$$y_k = Cx_k + v_k \tag{4.71}$$

Overall, we can think of the effects of process noise and measurement noise on the system through the following diagram:



Above: Noises may be added to the state and output equations of the system.

As we can see in the graphic above, we can think of process and measurement noises as being *injected* into the system at different points, and disturbing both the state vector and the measurements, respectively.

Standard Noise Assumptions

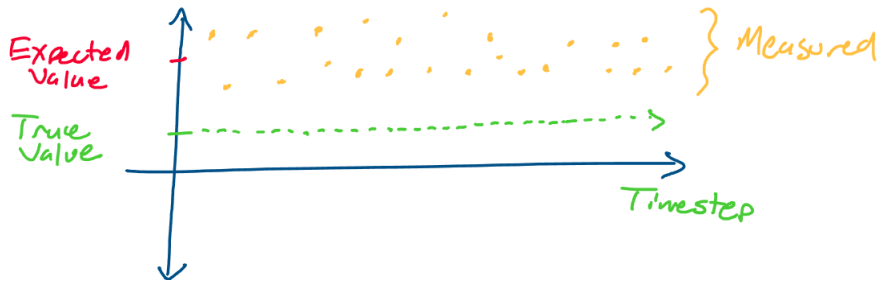
When describing the types of process and measurement noise applied to the system, there are several important properties we have to specify. Let's discuss some common assumptions we make when specifying process and measurement noise.

Firstly, we must specify the expected values of the process and measurement noises. Typically, we'll assume that both the process and measurement noise have zero expected value for all time:

$$\mu_{w_k} = \mathbb{E}[w_k] = 0 \in \mathbb{R}^n \tag{4.72}$$

$$\mu_{v_k} = \mathbb{E}[v_k] = 0 \in \mathbb{R}^p \tag{4.73}$$

Let's think about real-world systems, and reason about why zero expected value is a reasonable assumption for noise. Consider sensor noise, for instance. If the noise experienced by a sensor has *nonzero* expected value, then all of our measurements will effectively be *shifted* from the true value by a constant!



above: Noise with nonzero expected value will cause sensor readings to be offset from the true value.

In this case, all of our measurements will be offset by some constant, and our sensor will never accurately measure our system. We typically assume that our sensors are well-calibrated, so that there will be no constant offsets with respect to the true value of the system's state. This means that as long as our sensors are well-calibrated, we should assume zero expected value for noise.

What else do we need to specify? It's also important that we specify the auto-covariance of the process noise. Recall from the previous section that the auto-covariance of a random process tells us how the value of a random process at one instant in time is correlated with the value of the random process at another instant in time.

Typically, we'll assume that the value of the noise at a time k is *uncorrelated* with the value of the noise at a time j, as long as $j \neq k$. This means that the values of the noise at different instants in time have *no effect* on one another -

they're entirely random!
Furthermore, we'll assume that the auto-covariance of the noise with itself at the same instant in time does *not* depend on the index, k. This means that the relationships between each element in the noise vector are always the same, regardless of time.
Based on these assumptions, the measurement and process noise have auto-covariances defined as:

$$\Sigma_{w_k w_j} = \begin{cases} \Sigma_w & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases} \tag{4.74}$$

$$\Sigma_{v_k v_j} = \begin{cases} \Sigma_v & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases} \tag{4.75}$$

Where Σ_w and Σ_v are the covariances between the random vectors w_k and v_k between themselves at *any* instant in time.

$$\Sigma_w = \mathbb{E}[(w_k - \mu_{w_k})(w_k - \mu_{w_k})^T] \tag{4.76}$$

$$\Sigma_v = \mathbb{E}[(v_k - \mu_{v_k})(v_k - \mu_{v_k})^T] \tag{4.77}$$

Recall that our assumptions state that these covariances *do not* depend on k.
Why are these assumptions reasonable to make? Typically, when considering random noise experienced by sensors, the random noise experienced by the sensor at one instant in time does not influence the random noise experienced by the sensor at the next instant in time.
Furthermore, we assume that the actual characteristics of the sensor and the process noise largely remain constant with respect in time, which means that the covariances Σ_w and Σ_v should remain constant.
Since these two properties are such common assumptions to make, whenever random processes satisfy the conditions above on expectation and covariance, they are known as **white noise** random processes.
Now that we've discussed the auto-covariance of our noises, what about the cross-covariances? Typically, we assume that the noise experienced by the sensors and that random disturbances applied to the system have no relation to each other! We can express this mathematically by stating that the cross-covariance between the process noise w_k and the measurement noise v_k is zero at all times. Thus, we write:

$$\Sigma_{w_k v_j} = \mathbb{E}[(w_k - \mu_{w_k})(v_j - \mu_{v_j})^T] = 0 \tag{4.78}$$

In addition to assuming that our random processes will be white noise and uncorrelated, we'll also generally assume that their distributions at each point in time are Gaussian, as most sensor and process noises may be modeled as Gaussian processes. To express that w_k, v_k are white noise, Gaussian random processes, we may write:

$$w_k \sim \mathcal{N}(0, \Sigma_w) \in \mathbb{R}^n \tag{4.79}$$

$$v_k \sim \mathcal{N}(0, \Sigma_v) \in \mathbb{R}^p \tag{4.80}$$

Where 0 refers to the mean of the random process and Σ_w, Σ_v refer to the covariances. Note that we may use this notation for all k since we assume Σ_w, Σ_v are constant in time.

Stochastic Initial Conditions

Earlier, when discussing the evolution of deterministic (non-noisy) discrete time systems, we required an initial condition, x_0, to completely describe the states the system travels to in time.
Once we had the initial condition, we could repeatedly apply the state equation to figure out where the system would go. For instance, we could compute x_1 by calculating:

$$x_1 = Ax_0 + Bu_0 \tag{4.81}$$

And so on for $x_2, x_3, ..., x_n$. How might we perform a similar calculation for a stochastic system with process noise? Since the system experiences process noise at all point in time, specifying a *deterministic* initial condition is no longer sufficient!
Now, instead of providing a deterministic initial condition x_0, we must specify a probability distribution for x_0. In particular, it'll be important for us to know the expected value of x_0 and the auto-covariance of x_0 with itself at time 0.

$$\mu_0 = \mathbb{E}[x_0] \tag{4.82}$$
$$\Sigma_0 = \mathbb{E}[(x_0 - \mu_0)(x_0 - \mu_0)^T] \tag{4.83}$$

Once we know these properties and the probability distribution of the initial condition, we'll be ready to take on problems in state estimation.

4.2.2 The Kalman Filter

In this section, we'll finally propose a solution to the problem of state estimation that we've been building towards. Let's state our goal in state estimation a little bit more precisely. Imagine that we have a linear stochastic system with measurement and process noise that can be described by uncorrelated Gaussian white noise processes, w_k and v_k:

$$x_{k+1} = Ax_k + Bu_k + w_k,\ w_k \sim \mathcal{N}(0, \Sigma_w) \tag{4.84}$$
$$y_k = Cx_k + v_k,\ v_k \sim \mathcal{N}(0, \Sigma_v) \tag{4.85}$$

Suppose we have a sensor that gives us a sequence of measurements at each time step up until time k:

$$\{y_0, y_1, .., y_k\} \tag{4.86}$$

Where these measurements are taken according to the noisy output equation $y_k = Cx_k + v_k$. How can we use all of these noisy measurements to make the

best possible guess of our *entire* state vector at time k?
Let's discuss some of the challenges involved in this process. Firstly, we'll need to find some method of *filtering out* the noise from our measurements, despite the noise in the measurement being governed by a completely random white noise process.
Secondly, we want to reconstruct our *entire* state vector just from a set of partial measurements of the system! Recall that earlier, when discussing a turtlebot, we considered an example where our sensors only provided noisy measurements of x, y but our state vector contained x, y, ϕ. There may be *additional* quantities in the state vector that we can't directly measure but we'd still like to estimate. Let's frame this estimation problem more mathematically!

Least Squares Estimation

Let's try and frame the problem of optimal estimation of our state vector as an optimization problem. First, we'll define the variable $\hat{x}_k$ to represent our estimate of the state of the system at time k. At each time step k, what properties would we like our estimate $\hat{x}_k$ to have?
We'd like our estimate $\hat{x}_k$ to have the *smallest possible* error with respect to the true state of our system, x_k. Thus, we'd like our estimate $\hat{x}_k$ to minimize:

$$||x_k - \hat{x}_k||^2 \tag{4.87}$$

Note that we square the magnitude above so that we may more easily use techniques from linear algebra to solve our optimization problem. How can we frame this optimization problem to find $\hat{x}_k$ that minimizes $||x_k - \hat{x}_k||^2$?
Firstly, we notice that x_k is a *random variable*, as we have process noise in our system. Thus, we cannot directly minimize this quantity, as $||x_k - \hat{x}_k||^2$ has a random value! Therefore, instead of minimizing $||x_k - \hat{x}_k||^2$ directly, we seek to minimize the *expected value* of $||x_k - \hat{x}_k||^2$.
What else should we consider? When solving this optimization problem, we want to make use of as much information we know about our system as possible. Thus, we can use the measurements we've taken of the states $x_0, .., x_k$:

$$\{y_0, y_1, ..., y_k\} \tag{4.88}$$

To inform our estimate $\hat{x}_k$ of the system's state.
To make use of these measurements, instead of directly minimizing the expectation of $||x_k - \hat{x}_k||^2$, we can minimize the *conditional* expectation of $||x_k - \hat{x}_k||^2$ given that the measurements $y_0, ..., y_k$ have been taken of our system's state.
All together, we'd like to solve the following optimization problem for $\hat{x}_k^*$, our optimal estimate of the state at time k:

$$\hat{x}_k^* = \arg\min_{\hat{x}_k \in \mathbb{R}^n} \mathbb{E}[||x_k - \hat{x}_k||^2 | y_0, ..., y_k] \tag{4.89}$$

Where $|y_0, ..., y_k$ expresses the fact that the measurements $y_0, ..., y_k$ are given. By expanding the expression inside the expectation, we can prove that the

solution to this optimization problem is given by the following:

$$\hat{x}_k^* = \mathbb{E}[x_k|y_0, ..., y_k] \tag{4.90}$$

Thus, the best possible estimate of our system's state at time k is given by the *conditional expectation* of x_k given that the measurements $y_0, ..., y_k$ have been taken. Since this estimate minimizes the squared error, this solution is called the **least squares estimate** of the state vector.
For Gaussian process and measurement noises, we can miraculously solve for the conditional expectation, $\mathbb{E}[x_k|y_0, ..., y_k]$, in closed form. If we place all of our measurements $y_0, ..., y_k$ together into a large vector:

$$Y = [y_0^T, ..., y_k^T]^T \tag{4.91}$$

We can prove that we can solve for the conditional expectation $\mathbb{E}[x_k|y_0, ..., y_k] = \mathbb{E}[X_k|Y]$ using:

$$\mathbb{E}[x_k|Y] = \mathbb{E}[x_k] + \Sigma_{x_k Y}\Sigma_{YY}^{-1}(Y - \mathbb{E}[Y]) \tag{4.92}$$

Where $\Sigma_{x_k Y}$ is the covariance between x_k and Y and Σ_{YY} is the covariance between Y and itself.
Although this closed form solution *does* give us an optimal solution to the problem, is it the most convenient and computationally efficient solution? Let's think about some challenges that might come up with using this form of the least squares solution for robotic systems.
As time passes, Y, our vector of measurements, will become larger and larger. This means that the covariance matrix:

$$\Sigma_{YY} \in \mathbb{R}^{k \times k} \tag{4.93}$$

Will also become larger and larger. Examining the formula for conditional expectation from above, we notice that we must take the inverse of this matrix, Σ_{YY}^{-1}, when computing the least squares solution. Thus, after taking many measurements of this system, this matrix will become inefficient to invert!
How can we make use of all k measurements of the system while keeping an efficient solution? To answer this question, we'll develop a recursive state estimation method called *Kalman filter*.

Recursive Estimation with the Kalman Filter

To avoid having to invert this large matrix Σ_{YY} as time goes on, we may rewrite the solution to the least squares error estimation problem *recursively*. By deriving a recursive relationship between the system dynamics, noise properties, and expected values, we can efficiently compute the least squares estimate of a noisy system's state for any number of measurements.
The **Kalman filter** is the name of the set of equations that recursively compute the least squares solution to the estimation problem. For linear systems, since the Kalman filter equations provide the *least squares* solution, the Kalman filter

is provably the optimal least squares estimator.
Before we write out the equations of the Kalman filter, let's develop a little bit of notation to make our formulation of the filter cleaner. Firstly, let's introduce some notation for conditional expectation. For convenience, we'll write the conditional expectation of x_k given the measurements $y_0, ..., y_j$ as:

$$\mu_{k|j} = \mathbb{E}[x_k | y_0, ..., y_j] \tag{4.94}$$

We'll use similar notation to easily write the conditional covariance of x_k given measurements $y_0, ..., y_j$:

$$\Sigma_{k|j} = \mathbb{E}[(x_k - \mu_{k|j})(x_k - \mu_{k|j})^T] \tag{4.95}$$

This notation will prove convenient when writing out the estimation equations. In terms of this notation, we may define the Kalman filter as follows.

Theorem 7 ***Kalman Filter***
Given a system $x_{k+1} = Ax_k + Bu_k + w_k$, $y_k = Cx_k + v_k$, where w_k, v_k are uncorrelated, white noise random sequences, the following equations, which form the Kalman filter, give the best least squares estimate of the state vector.

$$\mu_{k+1|k} = A\mu_{k|k} + Bu_k \tag{4.96}$$
$$\Sigma_{k+1|k} = A\Sigma_{k|k}A^T + \Sigma_w \tag{4.97}$$
$$\mu_{k+1|k+1} = \mu_{k+1|k} + \Sigma_{k+1|k}C^T(C\Sigma_{k+1|k}C^T + \Sigma_v)^{-1}(y_{k+1} - C\mu_{k+1|k}) \tag{4.98}$$
$$\Sigma_{k+1|k+1} = \Sigma_{k+1|k} - \Sigma_{k+1|k}C^T(C\Sigma_{k+1|k}C^T + \Sigma_v)^{-1}C\Sigma_{k+1|k} \tag{4.99}$$

At each point in time, the estimate $\mu_{k|k}$ provides the best possible estimate of the state vector x_k using the measurements $y_0, ..., y_k$ with respect to the least squares distance to the actual value of x_k.
Let's review the role of each of these equations in recursively constructing the best estimate of the state. Let's begin with the first two equations:

$$\mu_{k+1|k} = A\mu_{k|k} + Bu_k \tag{4.100}$$
$$\Sigma_{k+1|k} = A\Sigma_{k|k}A^T + \Sigma_w \tag{4.101}$$

The first equation gives us a prediction of the state x_{k+1} given that we've taken measurements $y_0, ..., y_k$. To compute the estimate of the state at time $k+1$, this equation makes use of the state equation to predict where the system will be one time step in the future, starting from the estimated state $\mu_{k|k}$. Notice that this equation relies *entirely* on the system dynamics to predict the next state. Similarly, the second equation uses the system dynamics to predict the conditional covariance at the next time step.
Since these equations predict the future of the system based on the current

measurements, they are often known as the **predictor** equations.
Let's examine the differences between the first two state estimation equations and the second two. The second two equations state:

$$\mu_{k+1|k+1} = \mu_{k+1|k} + \Sigma_{k+1|k}C^T(C\Sigma_{k+1|k}C^T + \Sigma_v)^{-1}(y_{k+1} - C\mu_{k+1|k}) \quad (4.102)$$

$$\Sigma_{k+1|k+1} = \Sigma_{k+1|k} - \Sigma_{k+1|k}C^T(C\Sigma_{k+1|k}C^T + \Sigma_v)^{-1}C\Sigma_{k+1|k} \quad (4.103)$$

Let's examine the equation for $\mu_{k+1|k+1}$ first. As we can see, as opposed to purely using the system dynamics, like $\mu_{k+1|k}$, the equation for $\mu_{k+1|k+1}$ relies on y_{k+1}, the measurement taken at time $k+1$. Additionally, it uses the estimate $\mu_{k+1|k}$ that we made purely from the system dynamics. We find a similar comparison between the equations for $\Sigma_{k+1|k+1}$ and $\Sigma_{k+1|k}$.
Since the second two equations use the current measurement of the system to *correct* the estimate we made purely from system dynamics, they are often referred to as the **corrector** equations.
Through a continual process of *predicting* using the system dynamics and *correcting* using the latest measurement of the system, we're able to gain the least squares estimate of our system's state in an entirely recursive manner. Notice how the number of calculations required in each step *never* changes, unlike in our simple least squares formula!
Let's summarize how we may apply the Kalman filter to linear systems with a step-by-step process.

Definition 54 ***Kalman Filter Algorithm***
To optimally estimate the state of a linear system using the Kalman filter, apply the following procedure.

1. *Initialize the filter with $\mu_{0|-1} = \mathbb{E}[x_0]$ and $\Sigma_{0|-1} = \Sigma_0$, where $\mathbb{E}[x_0]$ and Σ_0 are provided by the probability distribution of the initial condition. Note that the -1 signifies that no measurements have been taken yet.*

2. *Take a first measurement y_0 of the system at time $k = 0$.*

3. *Apply the corrector equations to find $\mu_{0|0}$ and $\Sigma_{0|0}$, the state estimate and covariance using the first measurement:*

$$\mu_{0|0} = \mu_{0|-1} + \Sigma_{0|-1}C^T(C\Sigma_{0|-1}C^T + \Sigma_v)^{-1}(y_0 - C\mu_{0|-1}) \quad (4.104)$$

$$\Sigma_{0|0} = \Sigma_{0|-1} - \Sigma_{0|-1}C^T(C\Sigma_{0|-1}C^T + \Sigma_v)^{-1}C\Sigma_{0|-1} \quad (4.105)$$

4. *Apply the predictor equations to predict the state of the system and the covariance one step in the future:*

$$\mu_{1|0} = A\mu_{0|0} + Bu_0 \quad (4.106)$$

$$\Sigma_{1|0} = A\Sigma_{0|0}A^T + \Sigma_w \quad (4.107)$$

Where u_0 is the first input sent to the system.

5. *Take a measurement y_1 of the system at time $k = 1$.*
6. *Apply the corrector equations to find $\mu_{1|1}$ and $\Sigma_{1|1}$, the state estimate and covariance using the first two measurements:*

$$\mu_{1|1} = \mu_{1|0} + \Sigma_{1|0}C^T(C\Sigma_{1|0}C^T + \Sigma_v)^{-1}(y_1 - C\mu_{1|0}) \tag{4.108}$$
$$\Sigma_{1|1} = \Sigma_{1|0} - \Sigma_{1|0}C^T(C\Sigma_{1|0}C^T + \Sigma_v)^{-1}C\Sigma_{1|0} \tag{4.109}$$

7. *Repeat steps 4-6 to continually find the best estimate $\mu_{k|k}$ of x_k.*

You're encouraged to try deriving the different update steps of the Kalman filter equations - linearity of expectation and the uncorrelation between the noise sequences w_k and v_k will be particularly useful in this derivation. For a full derivation of the equations and the different forms of the Kalman filter, *Optimal Control and Estimation* by Stengel is a helpful reference. For a derivation of a continuous time version of the Kalman filter, *Optimization-Based Control* by Murray is useful.

4.2.3 Observability

So far in our development of the Kalman filter, we've assumed that recovering the state vector x_k from a set of noisy measurements $y_0, ..., y_k$ is always possible. Is this a valid assumption, or are there some cases where we *can't* recover the entire state vector of the system? Consider the following linear system, which has no noise applied:

$$x_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k \tag{4.110}$$
$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k \tag{4.111}$$

Examining the output equation, we see that the only term in the output vector is x_{k1}, the first element of the state vector. Can we use the output measurements y_k of this system to fully reconstruct the values of x_k?
When we look at the state equation of the system, we see that the dynamics of x_{k1} and x_{k2} are entirely *decoupled* - there are no cross-terms that relate x_{k1} and x_{k2} at any points in time! Due to this lack of relation, we have *no way* to fully reconstruct the state vector x_k from the partial measurements $y_k = x_{k1}$ of the state vector. Is there some way we can check for a problem like this in an arbitrary system? Consider the following definition.

Definition 55 ***Observability***
A linear discrete time system is said to be observable at time k if there exists

some time $j \geq k$ such that knowing the inputs $u_k, u_{k+1}, ..., u_j$ and the outputs $y_k, y_{k+1}, ..., y_j$ is enough to recover the full state vector x_k at time k.

Thus, if a system is *observable*, we'll be able to determine the full state vector of the system just from our knowledge of inputs and measurements. Using this definition, we would say that the example above is an *unobservable* system, as we cannot determine x from y and u.
How can we determine if an arbitary linear system is observable or not?

Proposition 9 ***Linear Observability Matrix***
Consider a linear system $x_{k+1} = Ax_k + Bu_k$, $y_k = Cx_k + Bu_k$, where $x_k \in \mathbb{R}^n$. If the following matrix has rank n, the system is observable.

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{4.112}$$

This matrix is known as the observability matrix.

Using the observability matrix, we can tell if we'll be able to reconstruct the entire state vector of our system from partial measurements. Notice how the definition of the linear observability matrix closely mirrors the definition of the linear *controllability* matrix! This similarity, which appears in many other results in control, hints at an underlying duality between the control and estimation of dynamical systems.