TODD D. MURPHEY

ACTIVE ROBOT LEARNING

Contents

Introduction

Relevant Books:

Amazingly, there are none. As a result, you all are guinea pigs.

Classical Control and Learning versus Active Learning

AL In the classical, passive view of machine learning (or estimation/-filtering or system identification), data is acquired and then passed through an algorithm (e.g., some machine learning algorithm) to arrive at a *model*. In the setting of *active learning* the data is connected to actions taken (e.g., by a robot moving in space), and those actions are determined by a combination of the model and the learning process itself.

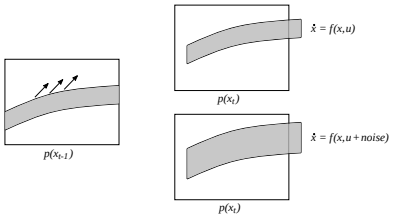


Figure 1: Classical passive learning and control versus active learning and control

Motivating Examples

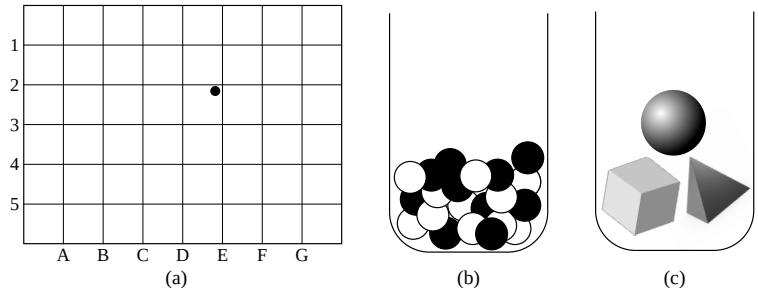


Figure 2: Active sensing scenarios. (a) what should your strategy be to get from somewhere in the environment to the dot? (b) How should you determine the number of dark marbles in a bag from only drawing one marble at a time? (c) How should you discriminate between objects in a bag?

Imagine that you need to use *beams* labeled 1-5 and A-G to get from the pictured initial condition to the final position shown in the figure. What is the right strategy. Without taking into consideration the sensor, you could just design a trajectory that is simply a line from the initial condition to the final location. But if there is uncertainty in the evolution, the beams (that give you labels 1-5 and A-G any time you cross them) provide extra information. Even more so, finding an intersection of two beams tells you *exactly* where you are, so finding the intersection of beam 2 and beam E would be particularly advantageous. From that location, the impact of uncertainty will be at its minimum because the target location is so close to the intersection location.

A simple example

Suppose you have a very simple system, like that seen in Fig 3. You can move in $\mathbb{R}^2$—up and down, right and left—and can take binary

measurements. These observations $o(t)$ are either "0" when the region is light or "1" when the region is dark. Can you make it to the goal state, noted by the $*$ in the figure? Here we have a state x governed by the differential equation $\dot{x} = u$ and a measurement model that looks like $Y = \begin{cases} 1 \text{ if } x \in dark \\ 0 \text{ if } x \in light \end{cases}$. Probably one should expect that this model is slightly uncertain, so adding noise to the model could make sense.

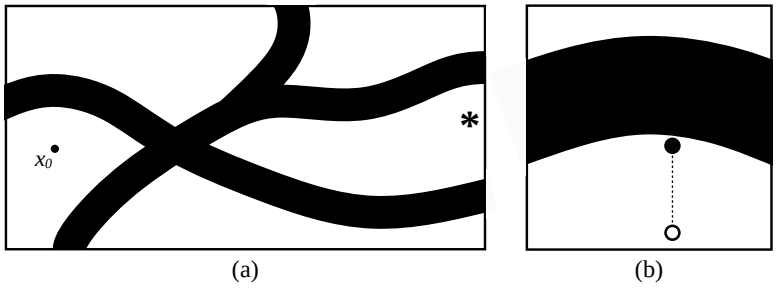


Figure 3: Imagine traversing a map with a binary sensor that can only read "0" in light areas and "1" in dark areas. This is likely the most simple sensor imaginable. How would you use it to get to the target $*$?

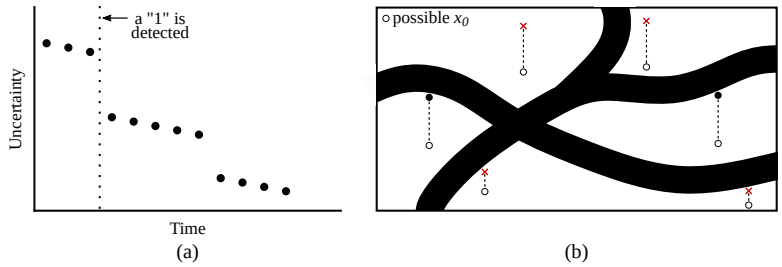


Figure 4: When the robot is moving—for instance, moving up—it is taking measurements and becoming more certain of where it is. When it reads "0" the robot knows it cannot have started in a place right below a black line (e.g., the bottom right corner). It will eventually hit a dark region of the map and read a "1". When it does so, its certainty changes dramatically, because now it knows it can only be along the edge between light and dark *and* that it must have started a certain distance below a dark region.

How will the uncertainty evolve as a function of movement? Consider Fig. 3(b), where the robot starts in a light region and takes several steps "up". As it does so, it receives "0" after "0", confirming it was not in the region just below a shaded in region (e.g., the two possible initial points on the bottom of Fig. 4(b)). Then, eventually, it hits a dark region and suddenly it knows that it is along one of the bottom contours (thus eliminating the possibilities of starting at either of the initial points on the top of Fig. 4(b)). These precipitous jumps in uncertainty, seen in Fig. 4 are great for the robot. They potentially mean that the robot can ignore incremental probability updates in favor of looking for these large jumps in information.

Components of Active Sensing and Active Learning

1. States x that can be impacted by decisions u; these are functions of time.
2. Uncertain parameters θ that can be estimated through sensory observations o (also functions of time).

3. Sensors:
   (a) range sensors
   (b) cameras
   (c) touch sensors (binary contact sensor, pressure, temperature, shear force, texture)
   (d) exotic nonlinear sensors, like electrosense.
4. Dynamic models describing how the sensor can move.

$$\dot{x} = f(x, u) \;\; x(0) = x_0$$

   where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, and $f(\cdot, \cdot)$ is a vector field. This motion model typically comes from either a principle (e.g., Newton's laws $\sum F = ma$ or Kirchhoff's laws or some such principle) or from data.
5. Measurement model $Y(x)$ describing sensor physics and sensor noise. This sensor model can come from physical modeling or from data.
6. Uncertainty models. Where does the uncertainty come from?
   (a) Is it from noise (e.g., stochastic forcing, like Brownian motion)?
   (b) Is it from spatial uncertainty (e.g., occlusions).
   (c) Is there uncertainty in what you are trying to find (e.g., distractors that look like what you are looking for)?
   (d) Is there uncertainty about structure of the world (e.g., how many predators are trying to eat you)?
7. Filters. These update *beliefs* about the world, typically represented as parameters.
8. Transition models of the world. This is a way of modeling how the world might behave. For example, you might be tracking something, and your model of how it behaves will not be anywhere near as good
9. Information measures. These will be the measures used to drive decisions.
   (a) How would you measure the value of visiting a region to improve your own state?
   (b) How would you measure the value of visiting a region to improve your understanding of someone else's state?

These will form the outline of the class, as we cover control, estimation, information theory, and techniques in active learning.

Optimal Control: Dynamics and Direct Methods

To develop a theory of active learning, we need to be able to translate data about the world into actions. Our focus here will be methods that focus on optimality criteria for doing so. Optimality criteria are not the only possible criteria one could use to drive actions—one could, for instance, simply define a proportional response to state, such as is done in PID control of linear systems—but optimality conceptually extends to many situations and systems that other techniques cannot treat.

There are several pieces involved in specifying a problem through optimality-based principles. These include a description of the dynamics, an objective function that represents the goal and distinguishes between good and bad outcomes. The ability to take derivatives of the dynamics and the objective will play a critical role in simplifying the procedural aspects of optimizing a function.

Dynamic Models

Typically a robotic system can be described by an ordinary differential equation.

$$\dot{x} = f(x, u) \quad x(0) = x_0 \tag{1}$$

In this equation we have the state $x \in \mathbb{R}^n$, the control $u \in \mathbb{R}^m$, and assume that f is at least differentiable with respect to both x and u.

Examples include:

1. $\dot{x} = u$, the single integrator system, where $x \in \mathbb{R}$ and $u \in \mathbb{R}$.
2. $\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$, the double integrator system $\ddot{x} = u$.
3. $\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) u_1 \\ \sin(\theta) u_1 \\ u_2 \end{bmatrix}$, the differential drive vehicle.

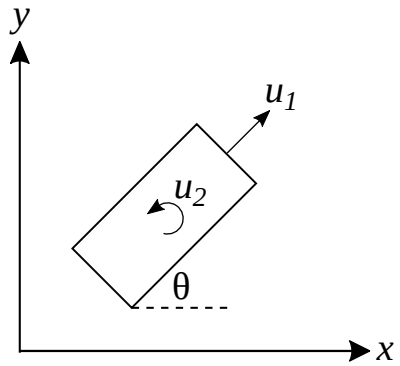


Figure 5: A vehicle capable of turning in place and moving forward

Note that sometimes variables like x will be used to denote vector or components of vectors. This will generally not lead to confusion, even when we say things like $x = \begin{bmatrix} x \\ y \end{bmatrix}$.

Objective functions

Objective functions generally come in a standardized form. We will see later that this standardized form is not always what we are looking for, but this provides a basis for thinking about optimization.

$$J(x(t), u(t)) = \int_0^T \ell(x(t), u(t))dt + m(x(T)) \tag{2}$$

Like the dynamic equations of motion, we will assume that $\ell(\cdot, \cdot)$ is at least differentiable with respect to $x(t)$ and $u(t)$.

Example of an objective function:

$$J(x(t), u(t)) = \int_0^T \frac{1}{2}(x(t) - x_d(t))^T Q(x(t) - x_d(t)) + \frac{1}{2}u(t)^T R u(t)dt + \frac{1}{2}(x(T) - x_d(T))^T P_1(x(T) - x_d(T))$$

where $Q \geq 0$, $R > 0$, $P_1 \geq 0$. (That is, these are matrices and they are either positive semi-definite or positive definite, meaning that when their arguments are nonzero they either return a positive number or, in the case of semi-definiteness, can also return zero. For instance, we could have $Q = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}$, $R = [1]$, and $P_1 = \begin{bmatrix} 10^3 & 0 \\ 0 & 10 \end{bmatrix}$ for the double integrator dynamics above.

Our goal is to optimize J, only using *trajectories* of Eq.(1). This optimization over a curve $u(t)$ or the pair $(x(t), u(t))$ is an *infinite dimensional* optimization.

Taking Derivatives: Df and ∇f

There are two notions of derivative, the Frechét and Gateaux derivatives. They both assume that states and the values of objective functions are elements of *vector spaces*—sets where addition and scalar multiplication make sense. Moreover, differentiation outside of the scalar-valued case requires a norm to be well defined, as we see momentarily.

Definition 1. A mapping $f : X \to Y$ (X and Y both vector spaces) is (Frechét) differentiable at $x_0 \in X$ if there is a continuous linear mapping $Df(x_0) : X \to Y$ such that

$$\lim_{\|z\|_X \to 0} \frac{\|f(x_0 + z) - f(x_0) - Df(x_0) \cdot z\|_Y}{\|z\|_X} = 0.$$

Note that in this definition we have $x_0 + z, x_0 \in X$ and $f(\cdot)$ of these quantities are in Y.

To make sense of this notion of derivative, let's go back to what we learned in introductory calculus of one variable:

$$\frac{df}{dx}(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$

Now, rearranging terms by moving the left hand side of the definition over to the right hand side, we get

$$0 = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0) - \frac{df}{dx}(x_0) \cdot h}{h}.$$

Why do we need the norms? Without the norms, we do not know what it means to "divide by" the denominator h. Even if we did, it would mean the definition would depend on the particular value h takes on. Instead, we require that the numerator and denominator be replaced by the norms of those elements of the vector space.

In both these definitions, $\frac{df}{dx}(x_0)$ is said to *approximate* f to first-order at x_0.

Definition 2. Another notion of derivative is

$$Df(x) \cdot h = \frac{df}{dx}(x) \cdot h = \frac{d}{d\varepsilon}|_{\varepsilon=0} f(x + \varepsilon z) \quad z \in X.$$

This is a weaker notion of derivative and is called the *Gateaux* derivative. However, this is the notion of derivative that we will use in practice because it is procedurally easy to work with. Any Frechét derivative that exists can be evaluated using the Gateaux derivative. However, a Gateaux derivative existing does not imply that the Frechét derivative exists.

EXAMPLE 1. Let's take the derivative using the Gateaux derivative. Assume that $f : \mathbb{R}^2 \to \mathbb{R}^2$

$$f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} x^2 \\ x + y^2 \end{bmatrix}.$$

You can probably guess that

$$Df\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 2x & 0 \\ 1 & 2y \end{bmatrix}.$$

However, let us find the derivative using the Gateaux derivative. Then $\frac{d}{d\varepsilon}|_{\varepsilon=0} f(x + \epsilon z)$ (where I am using x and z as vectors, purely for notational convenience) gives us

$$\begin{aligned}\frac{d}{d\varepsilon} f\left(\begin{bmatrix} x + \epsilon w \\ y + \epsilon z \end{bmatrix}\right)|_{\varepsilon=0} &= \frac{d}{d\varepsilon}\begin{bmatrix} (x + \epsilon w)^2 \\ (x + \epsilon w) + (y + \epsilon z)^2 \end{bmatrix}|_{\varepsilon=0} = \begin{bmatrix} 2xw + 2\epsilon w^2 \\ w + 2yz + 2\epsilon z^2 \end{bmatrix}|_{\varepsilon=0} \\ &= \begin{bmatrix} 2xw \\ w + 2yz \end{bmatrix} = \begin{bmatrix} 2x & 0 \\ 1 & 2y \end{bmatrix}\begin{bmatrix} w \\ z \end{bmatrix}\end{aligned}$$

Or, equivalently, using vector notion we can obtain the same result.

$$\frac{d}{d\varepsilon} f\left(\begin{bmatrix} x \\ y \end{bmatrix} + \epsilon \begin{bmatrix} w \\ z \end{bmatrix}\right)|_{\varepsilon=0} = Df\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) \cdot \begin{bmatrix} w \\ z \end{bmatrix}|_{\varepsilon=0} = \begin{bmatrix} 2x & 0 \\ 1 & 2y \end{bmatrix}\begin{bmatrix} w \\ z \end{bmatrix}$$

Moreover, we know that the derivative allows us to locally approximate the function.

$$f\left(\begin{bmatrix} x \\ y \end{bmatrix} + \epsilon \begin{bmatrix} w \\ z \end{bmatrix}\right) \approx f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) + \epsilon \begin{bmatrix} 2x & 0 \\ 1 & 2y \end{bmatrix}\begin{bmatrix} w \\ z \end{bmatrix}$$

EXAMPLE 2. Now, suppose that $J(x(t)) = \frac{1}{2}\int_0^1 x(t)^2 dt$, where $x(t) \in \mathbb{R}$. How do we differentiate J with respect to $x(t)$? Choose an arbitrary perturbation $z(t)$ and evaluate $\frac{d}{d\varepsilon}|_{\varepsilon=0} J(x(t) + \epsilon z(t))$.

$$\frac{d}{d\varepsilon} J(x(t) + \epsilon z(t))|_{\varepsilon=0} = \frac{1}{2}\int_0^1 \frac{d}{d\varepsilon}(x(t) + \epsilon z(t))^2 dt|_{\varepsilon=0} = \frac{1}{2}\int_0^1 (2x(t)z(t) + 2\epsilon z(t)^2) dt|_{\varepsilon=0} = \int_0^1 x(t)z(t)dt$$

EXAMPLE 3. Suppose that $f(x, u)$ is the right hand side of the equations of motion for the differential drive car. What does the derivative of those equations look like? Hint:

$$Df(x, u) \circ (\delta x, \delta u) = D_1 f(x, u)\delta x + D_2 f(x, u)\delta u,$$

where $D_i f$ is the derivative of the function f with respect to its i^{th} component—called the *slot derivative*.

Note that the gradient $\nabla f(x)$ is defined by the inner product (i.e., the dot product) in the following way.

$$Df(x) \cdot z = \langle \nabla f(x), z \rangle$$

This *defines* the gradient, so for every inner product, one gets a new gradient. This is not something to get too worried about for our purposes, but it does help make more sense out of the fact that the gradient $\nabla f(x)$ is *added* to the vector x, implying they belong to the same vector space.

Direct Methods in Optimal Control

Assuming that both an objective function and the dynamics are differentiable, one common approach to computing an optimal $u(t)$ is to use constrained finite dimensional optimization to approximate the optimizer. There are very good—or at least pretty good—optimization tools out there for constrained finite dimensional optimization (e.g., MATLAB's fmincon(), SNOPT, and others). These tools assume a problem of the form

$$\min_x h(x) \text{ such that } g(x) = 0 \tag{3}$$

where both h and g are assumed to be differentiable with respect to x (and often times you need to provide the software with these derivatives). Moreover, depending on the method used, inequality constraints can often be imposed as well (e.g., $g(x) \geq 0$); these can be used to indicate input saturation or unsafe parts of the state.

How can we obtain such a finite dimensional description of the optimization problem? Our approach will be to discretize both ℓ and f using quadrature to obtain something in the form of Eq. (3). For instance, suppose that we use the definition of the derivative to

discretize the continuous dynamics, splitting the time interval $[0, T]$ up into N different pieces.

$$\dot{x} = f(x, u) \implies \frac{x(t_{i+1}) - x(t_i)}{dt} \approx f(x(t_i), u(t_i))$$

which in turn implies the discrete time update law

$$x(t_{i+1}) = x(t_i) + dt f(x(t_i), u(t_i)).$$

Note that this equation holds for every time t_i, so there are N of these equations, all of which are constraints between x and u at various times. Also, note that $x(t_0) = x_0$.

Moreover, we can discretize J using Riemann integration approximation of the integral.

$$J = \int_0^T \ell(x(t), u(t))dt + m(x(T)) \approx \sum_{i=0}^{N} \ell(x(t_i), u(t_i))dt + m(x(t_N))$$

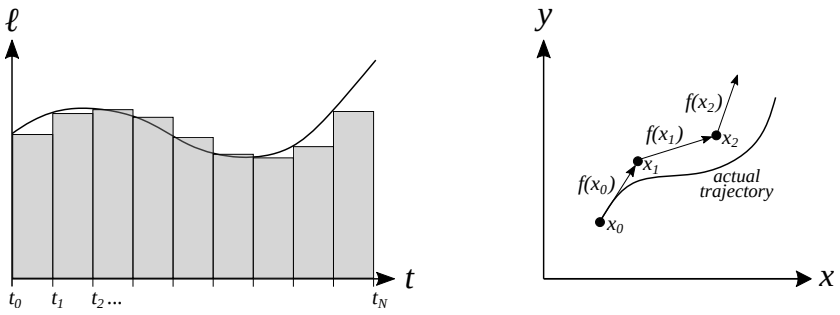


Figure 6: Both the objective function $\ell(\cdot)$ and dynamics can be discretized to create a finite dimensional optimization appropriate for direct methods. For instance, on the left is shown a Riemann sum approximation of the integral of ℓ, where the rectangles approximate the area under the curve. On the right, a trajectory is approximated with a discrete-time solver, such as Euler integration.

If we set

$$h(x(t_1) : x(t_N); u(t_0) : u(t_{N-1})) = \sum_{i=0}^{N} \ell(x(t_i), u(t_i))dt + m(x(T))$$

and set the constraints

$$g(x(t_1) : x(t_N); u(t_0) : u(t_{N-1})) = x(t_{i+1}) - x(t_i) - dt f(x(t_i), u(t_i)) \ \forall i$$

then we have a problem of the needed form.

A few notes on direct methods

First, *any* quadrature/interpolation scheme can be used to generate a finite dimensional description. The one I use above is Euler integration and Riemann integration of integrals, but one could use any Runge-Kutta/implicit Euler/midpoint/etc scheme for the dynamics; one could use any integration (e.g., trapezoidal rule, Simpson's rule,

et cetera) for the objective function. Moreover, higher-order representations of both will be better for numerical purposes.

Second, note that one could impose the terminal goal at time T as a constraint. For instance, one could say that $x(t_N) = x_d(T)$ in the list of constraints. If one gets a solution, this can be great. However, problems can become ill-conditioned, particularly if $x_d(T)$ is somehow infeasible.

Exercise

Take the double integrator system, and using the Q, R, P_1 from above, implement (using fmincon or something similar) a direct optimization with $N = 10, 10^2, 10^3$. Run each solution through a continuous simulation of the system. How well do the optimized controls work? What happens if you change the constraints to implicit Euler or the midpoint rule?

Summing Up

The use of optimization as a model of how to go from environmental estimates to control actions has a long history of being very successful. We will want to find objective functions appropriate for active learning. We should expect that we will want them to be differentiable, based on what we have discussed here.

Integral Equations and Nonlinear Iterative Optimization

In this lecture we largely will talk about derivatives of integral equations so that we can apply chain rule to an objective function $J(x(t), u(t))$. First, we need to be able to represent an ordinary differential equation as an integral equation, to make it easier to differentiate with respect to state and control variables.

Differential and integral equations

Let us remind ourselves of some things. Suppose we have a system such as:

$$\dot{x} = f(x(t), u(t)) \quad x(0) = x_0.$$

(We will typically suppress the dependence on t to avoid too much notation.) Then we know that an equivalent statement is that

$$x(t) = x_0 + \int_0^t f(x(\tau), u(\tau))d\tau.$$

This latter equation is the *integral* form of the differential equation. To see that the two are equivalent, note that we can check if $x(t)$ satisfies the differential equation. By the Leibniz rule we know that

$$\frac{d}{dt}\left[x_0 + \int_0^t f(x(\tau), u(\tau))d\tau\right] = f(x(t), u(t)).$$

So $x(t)$ satisfies the differential equation. Note that the differential equation incorporates the initial condition nicely.

This *integral representation* of the ordinary differential equation is now much easier to differentiate, specifically with respect to u. As a result, we can apply chain rule to a cost function that depends on $x(t)$ and $u(t)$ and evaluate the first derivative of $J(x(t), u(t)$ with respect to $u(t)$. When that derivative is equal to zero (for all possible directions $v(t)$), we know that we have a local minimizer, a local maximizer, or possibly a saddle point.

Derivatives of Objective Functions with Dynamic Constraints

Let us say we have an optimal control system where we cannot globally optimize the system by setting the derivative equal to zero. We might want to approach this problem by taking something along the lines of a gradient and then do gradient descent.[1] This would mean that our cost function looked like:

$$J = \int_0^T \ell(x, u)dt$$

and that we were minimizing J with respect to u subject to the constraint[2]

[1] In fact, taking a second-order approach can be even better, but for the purposes of these notes we will focus on first-order methods similar to gradient descent.

[2] What we should minimize with respect to is a key decision; we could also minimize with respect to the pair (x, u) that satisfies the dynamics, or, possibly, the pair (α, μ) that does not satisfy the dynamics—we will discuss this at length in the next lecture. We could also minimize solely with respect to u, which is also a common choice.

$$\dot{x} = f(x,u) \quad x(0) = x_0.$$

(Oftentimes ℓ will also depend explicitly on time t, but we will only include that later (e.g., when there is a time-varying reference trajectory); it does not impact the analysis.) To keep track of notation a bit more easily, we set notation $\xi = (x,u)$ and then differentiate $J(\xi) = J((x,u))$ in the direction $\zeta = (z,v)$. We then get

$$DJ(\xi)\cdot\zeta = \int_0^T D\ell(\xi)\cdot\zeta dt,$$

where the Lagrangian ℓ is allowed to vary with time (such as if there is a reference trajectory). Just as in the finite dimensional case, discussed shortly where the gradient—and thus the descent direction—is *defined* by minimizing a quadratic model, we can now potentially *choose* a quadratic model and obtain a ζ as a minimizer of a quadratic model.

For instance, we can simply choose the quadratic model $g(\zeta)$ (yes, ζ, not ξ!) in the following way:

$$g(\zeta) = \int_0^T D_2\ell(t,\xi)\cdot\zeta dt + \frac{1}{2}\int_0^T \|\zeta\|^2 dt$$

where the norm on the space of ζ (we haven't really specified what space that is yet) is something we get to define. For instance, it could just be the Euclidean 2-norm (where $Q = I_{n\times n}$ and $R = I_{m\times m}$) or the weighted Euclidean 2-norm (where $Q = Q^T \geq 0$ and $R = R^T > 0$). The key thing is that we get to choose this quadratic model in whatever way is most useful for generating the descent direction.

We have not yet discussed what happens to the constraint, but you can probably imagine that since we are only *locally* optimizing the cost function using the quadratic model, the *linearization* might be an appropriate way of representing the constraint. In this case, the quadratic model optimal control problem ends up simply being a standard optimal control problem! *This is the key that makes these techniques work, because we have a globally optimal method for optimizing linear quadratic cost functions with linear dynamic constraints.*

How can we see that the constraint involves the linearization? Assume that $\xi(t)$ is such that $x(t)$ and $u(t)$ are related through the differential equation. That means that $x(t) = x_0 + \int_0^t f(\xi(\tau))d\tau$. Hence,

$$\xi(t) = \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} = \begin{bmatrix} x_0 + \int_0^t f(\xi(\tau))d\tau \\ u(t) \end{bmatrix}$$

which implies (by taking the derivative of both sides with respect to ζ)

$$D\xi(t)\cdot\zeta(t) = \frac{d}{d\epsilon}\begin{bmatrix} x(t)+\epsilon z(t) \\ u(t)+\epsilon v(t) \end{bmatrix}_{\epsilon=0} = \frac{d}{d\epsilon}\begin{bmatrix} x_0+\epsilon z_0 + \int_0^t f(\xi(\tau)+\epsilon\zeta(\tau))d\tau \\ u(t)+\epsilon v(t) \end{bmatrix}_{\epsilon=0}$$

which in turn implies

$$D\xi(t)\cdot\zeta(t) = \begin{bmatrix} z(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} \overbrace{D_1 f(x(\tau),u(\tau))}^{A(\tau)} z(\tau) + \overbrace{D_2 f(x(\tau),u(\tau))}^{B(\tau)} v(\tau) \\ z_0 + \int_0^t \overbrace{Df(\xi(\tau))\cdot\zeta(\tau)}^{v(t)} \, d\tau \end{bmatrix}.$$

Note that this implies that $z(t)$ satisfies the differential equation

$$\dot{z} = A(t)z + B(t)v = D_1 f(x,u)z + D_2 f(x,u)v$$

such that $z(0) = z_0$.

Note that this derivation only requires that differentiating both sides of the integral equation $x(t) = x_0 + \int_0^t f(x(\tau),u(\tau))d\tau$ be a valid mathematical operation. Clearly one can differentiate both sides of an equality and still have equality; that is all we are taking advantage of here.

It is also important to recognize that the optimization is over ζ, *not* over ξ. (This hopefully reminds you of gradient descent in finite dimensions.) Hence, the $D\ell(\xi)$ is *fixed*–it is just some time-varying signal in the optimization problem. How do we optimize with respect to ζ? This turns out to be *very* similar to gradient descent in finite dimensions, but one has to interpret gradient descent appropriately.

Gradient Descent in Finite Dimensions

The fact that the above always depends on trajectories $(x(t),u(t))$, and is therefore infinite dimensional, often makes it challenging to have intuition about the optimization and what it means. To get better intuition, I will draw a (very solid) analogy to the finite dimensional setting of gradient descent.

The method of gradient descent is the algorithm most of us learned (in an ad-hoc way) to extremize functions. The basic idea is to find a direction that points "downhill" and then take a step in that direction. This is formalized in the following algorithm.

- Given initial data
- For $i = 0, 1, \ldots$
  - Determine a descent direction

    $$z_i = -\nabla f(x_i)$$

    One doesn't *need* to make this choice, but z_i should be capable of sufficient decrease.
  - Determine a step size (aka step length)

    $$\gamma_i = \arg\min_{\gamma > 0} f(x_i + \gamma z_i)$$

    One does not need to make this particular choice either.
  - update

    $$x_{i+1} = x_i + \gamma_i z_i$$

  - Repeat as needed

This algorithm will construct a sequence $\{x_i\}_{i=0}^{\infty}$ with $f(x_{i+1}) < f(x_i)\ \forall\ i$. What makes this complicated is illustrated below in the figure. A descent direction does not need to point directly at the local minimum. In particular, it will generally point in a direction that leads to the cost going *up* for a large enough step size.

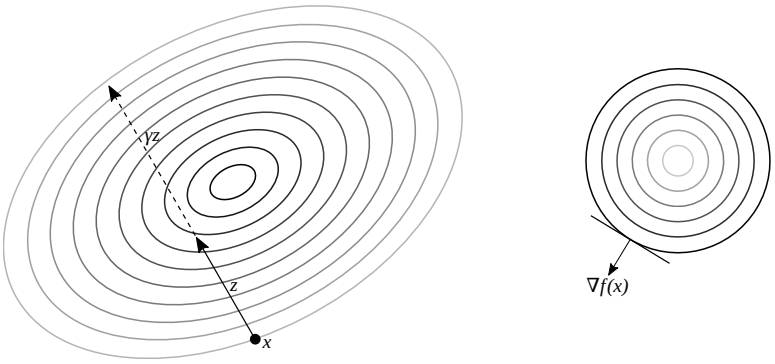


Figure 7: Left: Level set with descent direction and step size (from line search) included. Right: Gradient shown, perpendicular to level sets.

What questions can we ask about the gradient descent algorithm? If we employ gradient descent, are we guaranteed to descend? To converge? To meet necessary conditions of optimility if we do successfully converge? These are the fundamental questions that have to be answered, and it turns out that a relatively simple algorithm has guarantees on all these questions.

Now, where does gradient descent come from? That is, how does one obtain $z_i = -\nabla f(x_i)$? We look at the relationship between the

derivative and the gradient to find out. The gradient is *defined* by the following relationship.

$$Df(x) \cdot z = \langle \nabla f(x), z \rangle$$

Hence, one might want to maximize the amount of change in f one gets by choosing the right direction z, subject to not using too large of a z (modeled by a quadratic term).

$$\min_z Df(x) \cdot z + \frac{1}{2}||z||^2$$

We can rewrite this as

$$\min_z \langle \nabla f(x), z \rangle + \langle z, z \rangle$$

and then minimize by taking the derivative with respect to a perturbation w to z (not to x) and setting the result equal to zero.

$$\langle \nabla f(x), w \rangle + \langle z, w \rangle = \langle \nabla f(x) + z, w \rangle = 0 \quad \forall w$$

which implies that

$$z = -\nabla f(x).$$

The analog to finite dimensional gradient descent uses the descent direction from the equation shown earlier, and if we can solve for $(z(t), v(t))$ by doing that minimization, we can potentially do minimization in a way that directly mimics gradient descent in finite dimensions. That iterative process is illustrated in Fig. 8 for the first iterate of an iterative scheme.

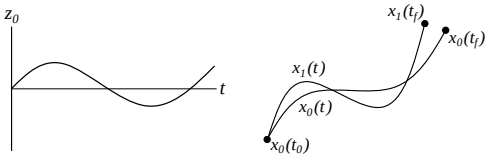


Figure 8: A trajectory $x_0(t)$ and a perturbation of the trajectory $x_1(t) = x_0(t) + \gamma z_0(t)$.

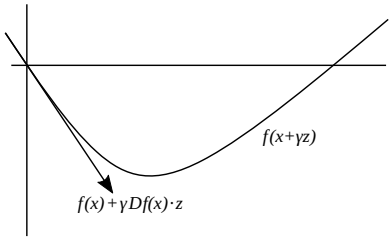


Figure 9: The Armijo line search

Armijo line searches

Let's go back and consider the map from $\gamma \mapsto f(x + \gamma z)$. We can approximate this map using different expansions of $f(\cdot)$. In particular, we can construct local quadratic models for determing the step size if we wish.

We must ensure that our step or update produces a sufficient decrease. Let's suppose that z_i is a descent direction for $f(\cdot)$ at x_i (i.e., $Df(x_i) \cdot z < 0$) (e.g., determined by a quadratic model). Suppose we want to avoid the situation where we converge *before* we get to an element of the vector space with zero derivative. This means we want to not only decrease at each step, we want to decrease *enough*. This is what Armijo realized.

The Armijo general sufficient decrease condition is:

$$f(x_i + \gamma z_i) \leq f(x_i) + \alpha \gamma Df(x_i) \cdot z_i \tag{4}$$

where $\alpha \in (0, 1)$ is some algorithmic parameter chosen by the designer.

The variable α is simply a constant chosen by the algorithm designer.[3] How should we choose γ? Armijo[4] says to use a sequence of form $\gamma = \beta^k$ $(k = 0, 1, 2, \ldots)$ until Eq.(4) is satisfied and take the smallest k that satisfies Eq.(4). Modifications of this basic algorithm are certainly possible. Also, note that β is just another algorithmic parameter.[5]

[3] For α a good choice can be $\alpha = 0.4$. Others, like Kelley, like $\alpha = 10^{-4}$.

[4] The Armijo step size rule is also sometimes known as backtracking line search.

[5] A good choice is $\beta = 0.7$.

Formal Properties of Armijo Line Searches

If you apply Armijo line searches, you will only converge to a point where the directional derivative is equal to zero.

Optimization Using Armijo

Given x_0, $f(\cdot)$, $Df(\cdot)$, $\alpha \in (0, \frac{1}{2})$, $\beta \in (0, 1)$, and maybe ϵ
While $\|\nabla f(x_i)\| > \epsilon$
Choose z_i by solving $z_i = \arg\min Df(x_i) \circ z + \langle z, z \rangle$
$n = 0$ $\gamma = \beta^n$
While $f(x_i + \gamma z_i) > f(x_i) + \alpha\gamma Df(x_i) \circ z_i$
$n = n + 1$
$\gamma = \beta^n$
end while
$x_{i+1} = x_i + \gamma z_i$
end while

Questions the student should be asking include the following. Does Armijo converge? Can we get an idea of how small a γ may be required? (Yes, in particular if we obtain z_i by minimizing a quadratic model function and even for general z_i.) Does Armijo guarantee that the algorithm converges to a point that satisfies some condition of optimality? Before getting to those questions we need to show that a γ even exists that satisfies Eq.(4) for γ small enough.

For formal results on Armijo line searches, including the fact that using an Armijo line guarantees convergence to a point that satisfies $Df(x) = 0$, see the terrific book *Iterative Methods for Optimization* by C.T. Kelley.

Examples

Consider the system $\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} \cos(x_2) \\ -\sin(x_1) + \cos(u) \end{bmatrix}$ What ordinary differential equation does a perturbation to a particulate $(x(t), u(t))$ look like?

$$A(t) = D_1 f(x, u) = \begin{bmatrix} 0 & -\sin(x_2) \\ -\cos(x_1) & 0 \end{bmatrix} \text{ and } B(t) = D_2 f(x, u) = \begin{bmatrix} 0 \\ -\sin(u) \end{bmatrix}$$

As a result, perturbations $z(t)$ must satisfy the differential equation $\dot{z} = A(t)z + B(t)v$ with $z(0) = z_0$, where z_0 and $v(t)$ parameterize all possible perturbations.

State Transition, Convolution, and Riccati Equations

State Transition Matrices

Consider the linear time-varying system

$$\dot{x} = A(t)x(t) \qquad x(t_0) = x_0.$$

The state transition matrix for this system is the matrix $\Phi(t, t_0)$ that satisfies the *matrix-valued* differential equation

$$\dot{\Phi} = A(t)\Phi \qquad \Phi(t_0, t_0) = Id_{n\times n}.$$

Amazingly,

$$x(t) = \Phi(t, t_0)x_0$$

holds for any linear system. In the special case where $A(t) = A$—the system is linear time-invariant (LTI)—it turns out that $\Phi(t, t_0) = e^{A(t-t_0)}$, the matrix exponential of $A(t - t_0)$.

Convolution equations

Consider the linear time-varying system

$$\dot{x} = A(t)x(t) + B(t)u(t) \qquad x(t_0) = x_0.$$

This does not have the same linear structure as the previous system, so the state transition matrix associated with $A(t)$ is not quite enough to compute the solution. Instead, we get the *convolution* equation

$$x(t) = \Phi(t, t_0)x_0 + \int_{t_0}^{t} \Phi(t, \tau)B(\tau)u(\tau)d\tau.$$

Now, this equation is not quite as complex as it looks. Roughly speaking, it says that the control is coupled to what the system would have done based on its initial condition through the integral equation that weighs the input against the state transition matrix operating on the input. If $A = 0$, then we expect to be able to directly control the state; this is exactly what is predicted because for $A = 0$ we have $\Phi(t, t_0) = Id_{n\times n}\ \forall\ t$.

Using Linear Solutions to Find Descent Directions

Assume we have linear time-varying equations of motion. The dynamics are

$$\dot{x} = A(t)x + B(t)u \qquad x(0) = x_0$$

and

$$x(\tau) = \Phi(\tau, 0)x_0 + \int_0^{\tau} \Phi(\tau, s)B(s)u(s)ds.$$

Let's assume we have the simplest cost function imaginable–a cost that is quadratic in x and u:

$$J(u(\cdot)) = \frac{1}{2}\int_0^T x(t)^T Q(t)x(t) + u(t)^T R(t)u(t)dt + \frac{1}{2}x(T)^T P_1 x(T) \quad (5)$$

where $Q = Q^T \geq 0$, $R = R^T > 0$, $P_1 = P_1^T \geq 0$. We now formally differentiate $J(\cdot)$ with respect to $u(\cdot)$ (assuming, potentially incorrectly, that everything is nicely differentiable[6]).

[6] Note that we cannot possibly know whether or not things are differentiable because we do not know what space we are working in.

Calculation of Necessary Conditions for Optimality

$$\frac{d}{d\epsilon}J(u(t)+\epsilon v(t))|_{\epsilon=0}$$
$$= \int_0^T x^T Qz + u^T Rvdt + x(T)^T P_1 z(T)$$
$$\text{where } z(\cdot) = \frac{\partial x(\cdot)}{\partial u}$$
$$= \int_0^T \left[\underbrace{x(\tau)^T Q}_{a(\tau)^T} \underbrace{\left[\int_0^\tau \Phi(\tau,s)B(s)v(s)ds\right]}_{z(\tau)} + \underbrace{u(\tau)^T R(\tau)}_{b(\tau)^T} v(\tau) \right] d\tau + \underbrace{x(T)^T P_1}_{p_1^T} \int_0^T \Phi(T,s)B(s)v(s)ds$$
$$= \int_0^T a(\tau)^T \left[\int_0^\tau \Phi(\tau,s)B(s)v(s)ds\right] d\tau + p_1^T \int_0^T \Phi(T,s)B(s)v(s)ds + \int_0^T b(\tau)^T v(\tau)d\tau$$
$$= \int_0^T \left[\int_0^\tau a(\tau)^T \Phi(\tau,s)B(s)v(s)ds\right] d\tau + p_1^T \int_0^T \Phi(T,s)B(s)v(s)ds + \int_0^T b(\tau)^T v(\tau)d\tau$$

Now change the order of integration, noting in Fig. 10 the change in boundary conditions on the integrals.

$$= \int_0^T \left[\int_s^T a(\tau)^T \Phi(\tau,s)d\tau\right] B(s)v(s)ds + p_1^T \int_0^T \Phi(T,s)B(s)v(s)ds + \int_0^T b(\tau)^T v(\tau)d\tau$$
$$= \int_0^T \left[\underbrace{\int_s^T a(\tau)^T \Phi(\tau,s)d\tau + p_1^T \Phi(T,s)}_{p(s)^T} \right] B(s)v(s)ds + \int_0^T b(\tau)^T v(\tau)d\tau$$
$$= \int_0^T (p(\tau)^T B + b(\tau)^T)v(\tau)d\tau.$$

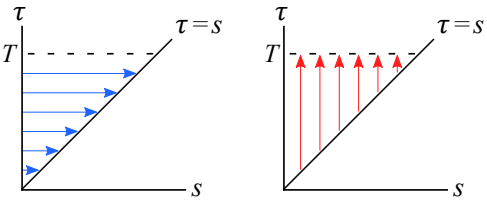


Figure 10: Changing order of integration.

Now, $p(t)$ (seen again below) as it is defined doesn't seem very easy to compute, but it turns out that it satisfies a nice but somewhat unexpected differential equation. You can think of the integral expression that defines $p(s)^T$ as being the convolution equation, except that in this case the state transition matrix $\Phi(\tau,s)$ operates on a boundary condition at the final time T instead of the initial time 0.

$$p(t)^T = \int_t^T a(\tau)^T \Phi(\tau,t)d\tau + p_1^T \Phi(T,t) \implies p(t) = \int_t^T \Phi(\tau,t)^T a(\tau)d\tau + \Phi(T,t)^T p_1$$

Moreover, integration is being performed on the first argument of the state transition matrix $\Phi(\tau, s)$ instead of the second argument. (In order to formally get the convolution equation, we would need to reverse this.[7]) Nevertheless, as a consequence of this integral definition of p, we find (in the exercises) that $p(\tau)$ satisfies the differential equation

$$\dot{p} = -A^T p - Qx \quad p(T) = p_1$$

and we get

$$DJ(u) \cdot v = \frac{d}{d\epsilon} J(u(t) + \epsilon v(t))|_{\epsilon=0} = \int_0^T [p(\tau)^T B + b(\tau)^T] v(t) dt$$

so, to be optimal, $u(\cdot)$ must satisfy the equation

$$[p(t)^T B + u(t)^T R] = 0$$

or, more conveniently,

$$B^T p(t) + Ru(t) = 0 \tag{6}$$

which implies

$$u = -R^{-1} B^T p(t).$$

Note that if we write down the three conditions for optimality together, we get

$$\begin{aligned} \dot{x} &= Ax + Bu \\ \dot{p} &= -A^T p - Qx \\ 0 &= B^T p(t) + Ru(t). \end{aligned}$$

This is the differential statement of the famous *Maximum Principle*, which we will discuss more later. It will turn out that there is a natural way to obtain these equations from an appropriately defined Hamiltonian; this will be very helpful to us later when we want to solve some LQR problems that have off-diagonal terms.

Rewriting the above equations by substituting in the value of u, we get

$$\begin{aligned} \dot{x} &= Ax - BR^{-1} B^T p(t) \\ \dot{p} &= -A^T p - Qx \\ 0 &= B^T p(t) + Ru(t). \end{aligned}$$

which, in matrix form, looks like

$$\begin{bmatrix} \dot{x} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} A & -BR^{-1}B^T \\ -Q & -A^T \end{bmatrix} \begin{bmatrix} x \\ p \end{bmatrix} \qquad \begin{bmatrix} x(0) \\ p(T) \end{bmatrix} = \begin{bmatrix} x_0 \\ p_1 \end{bmatrix} \tag{7}$$

[7] Let us use the fact that $\Phi(t, t_0) = \Phi^{-1}(t_0, t)$ to rewrite the equation to look more like the convolution equation $p(s) = \int_s^T a(\tau)^T \Phi^{-1}(s, \tau) d\tau + p_1^T \Phi^{-1}(s, T)$. This is *almost* the convolution equation for a system with Φ^{-1} as the state transition matrix. However, since p flows *backwards* in time from p_1 at time T to "final" time s (treating s like a terminal time), the integration must also be reversed, producing the convolution equation $p(s) = -\int_T^s a(\tau)^T \Phi^{-1}(s, \tau) d\tau + p_1^T \Phi^{-1}(s, T)$. Due to the properties of the state transition matrix, it turns out that if Φ is the state transition matrix for LTV systems with linearization $A(t)$, Φ^{-1} is the state transition matrix for a system with linearization $-A(t)$—so the convolution equation indicates $p(s)$ can be computed from a linear affine ordinary differential equation.

Note this does not have the same boundary condition structure that we are used to (e.g., initial value problems) because it has an *initial* condition in x and a final condition in p. This is called a two-point boundary value problem (TPBVP). You can just solve this in MATLAB, but we are going to opt to be more clever than that.

The first-order optimality condition is equivalent to the *solvability* of (7). The question is whether or not we can solve this TPBVP? It turns out that something magical happens here; in particular, it is true that $p(t) = P(t)x(t)$ for some choice of $P(t)$ (at least near $t = T$).

Riccati Equations

Now, what differential equation does $P(t)$ satisfy, assuming it exists at all? We want to know because we would like to be working with lower-dimensional, better-conditioned systems if possible. We want $P(\cdot)$ so that $p(t) = P(t)x(t)$ for $t < T$ (hopefully $t \in [0, T]$, but we will see in the next section that we can't necessarily guarantee that). What do we do? What we always do: Differentiate!

$$\begin{aligned} p &= Px \\ \Rightarrow \dot{p} &= \dot{P}x + P\dot{x} \\ &= \dot{P}x + P(Ax - BR^{-1}B^TPx) \\ \text{and } \dot{p} &= -Qx - A^TPx \\ \Rightarrow 0 &= (\dot{P} + PA + A^TP - PBR^{-1}B^TP + Q)x \end{aligned}$$

This equation has to hold for all possible trajectories $x(\cdot)$, so the matrix-valued differential equation

$$\dot{P} + PA + A^TP - PBR^{-1}B^TP + Q = 0 \qquad P(T) = P_1$$

must hold. This is a *Riccati equation*. This Riccati Equation is a quadratic equation in P that runs backward in time. Since it is smooth, the soution will exist and be unique from T down to some $t^* < T$ which implies that $P(t)$ on $(t^*, T]$ is well defined. We *hope* that $t^* < t_0$.

Exercises

Given a two dimensional linear system (e.g., $\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$) and a quadratic objective function (e.g., $J = \int_0^1 \frac{1}{2}x^Tx + u^2dt + 10x(T)^Tx(T)$), find the Riccati solution for the optimal control and numerically convince yourself that it is the optimizer. (Think about how to do this using tools we have discussed in class.)

Iterative Optimization, Maximum Principle, and Two Point Boundary Value Problems

The Maximum Principle

We have just shown a very, *very* limited version of what is called the Pontryagin Maximum Principle. We know that for the LQ problem

$$J = \int_0^T \frac{1}{2}x^T Q x + \frac{1}{2}u^T R u dt + \frac{1}{2}x(T)^T P_1 x(T)$$

such that

$$\dot{x} = A(t)x + B(t)u \quad x(0) = x_0,$$

we get

$$\begin{aligned} \dot{x} &= A(t)x + B(t)u & x(0) &= x_0 \\ \dot{p} &= -A(t)^T p - Q(t)x & p(T) &= p_1 \\ 0 &= B(t)^T p(t) + R(t)u. \end{aligned}$$

A notational restatement of this can be achieved by using the *Hamiltonian* formalism. Define

$$H = \ell + p^T f$$

where $\ell = \frac{1}{2}x^T Q x + \frac{1}{2}u^T R u$, p is the adjoint variable, and $f = A(t)x + B(t)u$. Then these equations are equivalent to Hamilton's equations

$$\begin{aligned} \dot{x} &= D_p H^T \\ \dot{p} &= -D_x H^T \\ 0 &= D_u H^T \end{aligned}$$

(We will also sometimes use the notation $D_p H = H_p$, $-D_x H = -H_x$, and $D_u H = H_u$.) For the case of the LQ problem, we have already shown that these equations provide the optimal control as a Two Point Boundary Value Problem. What is amazing is that these equations hold for *any* nonlinear system and for any objective function! This result, called the *Pontryagin Maximum Principle*, can be stated as the following.

Theorem 1. *Given $J = \int_0^T \ell(t, x(t), u(t))dt + m(x(T))$ and f sufficiently differentiable, and $H = \ell + p^T f$, the optimal control u satisfies the following relation.*

$$\begin{aligned} \dot{x} &= f(t, x(t), u(t)) & x(0) &= x_0 \\ \dot{p} &= -f_x(t, x(t), u(t))^T p - \ell_x(t, x(t), u(t)) & p(T) &= p_1 = m_x(T) \\ 0 &= f_u(t, x(t), u(t))^T p(t) + \ell_u(t, x(t), u(t)). \end{aligned}$$

We will use this fact many times in later sections.

Back to iterative optimization

This is basically the same thing that we did in the last section! That is, minimizing this linear quadratic problem is *very* similar to optimizing when there is a reference signal in the Lagrangian ℓ. To see this, define

$$(a(t)^T, b(t)^T) = D\ell(\xi).$$

Now, the Hamiltonian for the system with ζ as the optimization variable is

$$H = \underbrace{a^T z + b^T v}_{D\ell(\xi)\cdot\zeta} + \frac{1}{2}\underbrace{(z^T Q z + v^T R v) + p^T(Az + Bv)}_{||\zeta||^2}$$

(where Q and R are whatever we have chosen for the quadratic model) so that the optimality conditions become

$$\begin{aligned}
\dot{z} &= H_p^T = Az + Bv \\
\dot{p} &= -H_z^T = -a - Qz - A^T p \\
0 &= H_v^T = b + Rv + B^T p
\end{aligned}$$

so that $v = -R^{-1}(B^T p + b)$. The key here is that the a and b terms both typically depend on the *error* between the actual trajectory and the desired trajectory, so those terms will be responsible for locally driving the system cost down.

Assume that $p = Pz + r$ just like the last section. Taking derivatives with respect to time, we find that

$$\begin{aligned}
p &= Pz + r \\
\dot{p} &= \dot{P}z + P\dot{z} + \dot{r} \\
-a - Qz - A^T p &= \dot{P}z + PAz + PBv + \dot{r} \\
-a - Qz - A^T Pz - A^T r &= \dot{P}z + PAz + PBv + \dot{r} \\
-a - Qz - A^T Pz - A^T r &= \dot{P}z + PAz + PB(-R^{-1}(B^T p + b)) + \dot{r} \\
-a - Qz - A^T Pz - A^T r &= \dot{P}z + PAz - PBR^{-1}B^T p - PBR^{-1}b + \dot{r} \\
-a - Qz - A^T Pz - A^T r &= \dot{P}z + PAz - PBR^{-1}B^T Pz - PBR^{-1}B^T r - PBR^{-1}b + \dot{r} \\
0 &= (\dot{P} + PA + A^T P - PBR^{-1}B^T P + Q)z + (\dot{r} + A^T r - PBR^{-1}B^T r + a - PBR^{-1}b) \\
0 &= (\dot{P} + PA + A^T P - PBR^{-1}B^T P + Q)z + (\dot{r} + (A - BR^{-1}B^T P)^T r + a - PBR^{-1}b).
\end{aligned}$$

So we have two differential equations to solve:

$$\begin{aligned}
0 &= \dot{P} + PA + A^T P - PBR^{-1}B^T P + Q \\
0 &= \dot{r} + (A - BR^{-1}B^T P)^T r + a - PBR^{-1}b.
\end{aligned}$$

These are subject to terminal boundary conditions, of course. Remember, that we know what $p(T)$ is, and given z_0 we know what $z(T)$ is. As a consequence, we know what $P(T)$ and $r(T)$ must be as well.

We will obtain the *descent direction* defined by these equations by solving differential equations. What is the descent direction? It is $\zeta = (z, v)$ satisfying

$$\begin{aligned}\dot{z} &= Az + Bv \\ v &= -R^{-1}B^T Pz - R^{-1}B^T r - R^{-1}b\end{aligned}$$

where we have not yet defined what the initial condition of z should be. There are two "natural" choices of initial condition. One is that $z(0) = 0$, so that the perturbation at time zero is zero, making it easier to regulate at time $t = 0$. The other possibility is to choose an optimal $z(0)$—to minimize the objective with respect to z_0.

$$\min_{z_0} g(\zeta) = \min_{z_0} \int_0^T D_2\ell(t, \xi) \cdot \zeta + \frac{1}{2}\|\zeta\|^2 dt$$

The solution to this problem is $z(0) = -P^{-1}(0)r(0)$, but this isn't easy to show. The standard method involves computing the "cost to go" function; we will discuss this in the final section of these notes—the derivation of this extraordinarily compact result is quite involved—but for now all you really need to know is that $z(0) = -P^{-1}(0)r(0)$ is a good choice in your code.

The iLQR algorithm

To recap, last lecture we derived an optimal controller for a linear time-varying system. It was shown that a backwards propagating Riccati equation could be solved to obtain a optimal controller with respect to any quadratic cost:

Given the cost

$$J(u(\cdot)) = \frac{1}{2}\int_0^T x(t)^{\mathrm{T}} Q(t) x(t) + u(t)^{\mathrm{T}} R(t) u(t)\ \mathrm{d}t + \frac{1}{2} x(T)^{\mathrm{T}} P_1 x(T),$$

the optimal system behavior is given as

$$\begin{aligned}\dot{x}(t) &= (A(t) - B(t)K(t))x(t), \quad x(t_0) = x_0, \\ -\dot{P}(t) &= P(t)A(t) + A(t)^{\mathrm{T}}P(t) - P(t)B(t)R(t)^{-1}B(t)^{\mathrm{T}}P(t) + Q(t), \quad P(T) = P_1, \\ K(t) &= R(t)^{-1}B(t)^{\mathrm{T}}P(t).\end{aligned}$$

Then, in this lecture, the methodology was extended to the case when the system is governed by nonlinear dynamics. However, instead of deriving a globally optimal controller the optimal descent direction was obtained. That is, we found the best way to change

the current trajectory, $\xi = (x, u)$, to obtain a more optimal one. The descent direction, $\zeta = (z, v)$, is computed as:

Given the cost and nonlinear dynamics

$$J(u(\cdot)) = \int_0^T l(t, (x, u))\, \mathrm{d}t \quad \text{and} \quad \dot{x}(t) = f(x, u),$$

the descent direction, $\zeta = (z, v)$, is computed as

$$\begin{aligned}
\dot{z}(t) &= A(t)z(t) + B(t)v(t), \quad x(t_0) = x_0, \\
v(t) &= -R(t)^{-1}B(t)^{\mathrm{T}}P(t)z(t) - R(t)^{-1}B^{\mathrm{T}}(t)r(t) - R(t)^{-1}b(t), \\
-\dot{P}(t) &= P(t)A(t) + A(t)^{\mathrm{T}}P(t) - P(t)B(t)R(t)^{-1}B(t)^{\mathrm{T}}P(t) + Q(t), \\
-\dot{r}(t) &= (A - BR^{-1}B^{\mathrm{T}}P)^{\mathrm{T}}r + a - PBR^{-1}b.
\end{aligned}$$

where $(a(t)^{\mathrm{T}}, b(t)^{\mathrm{T}}) = D_2 l(t, \xi)$, $A(t) = D_1 f(x, u)$, and $B(t) = D_2 f(x, u)$.

A gradient descent algorithm can now be created based on this result.

Algorithm 1: Iterative Linear Quadratic Control

Data:
$\xi_0 = (x_0, u_0)$ Initial Trajectory (Initial Guess)
$J(\xi_0)$ Considered Cost
$\alpha = (0, \frac{1}{2})$, $\beta = (0, 1)$, and ϵ (a small number)
$i = 0$
while $||\zeta|| > \epsilon$ **do**
compute descent direction, $\zeta_i = (z_i, v_i)$
$\zeta_i = \arg\min_{\zeta} DJ(\xi_i) \circ \zeta + \frac{1}{2}||\zeta||^2$
compute line search using descent direction ζ_i
n = 0
while $J(\xi_i^+) > J(\xi_i) + \alpha\gamma DJ(\xi_i) \circ \zeta_i$ **do**
$u_i^+ = u_i + \gamma v_i$
$x_i^+ = x(t_0) + \int_0^T f(t, (x_i^+, u_i^+))\, \mathrm{d}t$
$\xi_i^+ = (x_i^+, u_i^+)$
$n = n + 1$
$\gamma = \beta^N$
end while
$\xi_{i+1} = \xi_i^+$
i = i + 1
end while

The algorithm is iterative since it successively updates the trajectory based on the computed descent direction. Note that the updated state trajectory, x_i^+, is not computed directly from the state descent direction, z_i. Instead, only the control signal is directly updated. Therefore, it cannot be assumed that the new state trajectory follows

the computed descent direction ($x_i^+ \neq x_i + \gamma z_i$). Indeed, $\xi_i^+ = \xi_i + \gamma\zeta$ may not even be a dynamically feasible trajectory. In the next couple of lectures projection operators are introduced to address this issue.

Solving Optimal Control Problems Using Two Point Boundary Value Problems

From the Maximum Principle, we can also obtain a numerics-based approach to solving an optimal control problem. That is, we can just try to solve the two point boundary value problem directly (e.g., MATLAB, Mathematica, python have built-in solvers). If the solver returns an answer, you have your solution!

Exercises

Show numerically that the solution to the Riccati equation (obtained by solving the matrix-valued ordinary differential equation) and the solution to the Two Point Boundary Value Problem (obtained using an appropriate numerical solver) are nearly, but not exactly, the same. Use the example from last lecture as the example.

Use the Armijo line search for a single iterate of iLQR.

Apply iLQR to the vehicle example, using a semi-circle as an initial trajectory and an infeasible straight line corresponding to parallel parking as a reference trajectory.

Probability

Much of this section—and the following sections on filtering—is modeled after the book *Probabilistic Robotics* by Thrun, Burgard, and Fox (2005). The goal here is to provide a brief introduction into reasoning about robots from the perspective of probabilities and, in my opinion more importantly, distributions and their role in capturing concrete uncertainties faced by robots. The first part of this section will introduce the definitions we need. The second part will try to connect these definitions to concrete problems in robotics and to give students a sense of what types of uncertainties we are good at dealing with and what types of uncertaintes we are not good at dealing with.

Basic Definitions

The most basic idea in probability is that of a *random variable* $\mathcal{X}$, which can take on particular values x. Often these values are part of the state of a robot (e.g., $x \in \mathbb{R}^n$), but they are just as often discrete outcomes (like the outcome of flipping a coin). As a result, we will need both discrete and continuous random variables. That is, either

$$x \in \{x_1, x_2, ..., x_n\} \text{ or } x \in S \subset \mathbb{R}^n.$$

The random variables take on values with a particular *probability* $p(X = x)$ (typically using notation $p(x)$ unless there is the potential for confusion). These probabilities must satisfy the following requirements.

$$\sum_x p(X = x) = 1$$
$$p(X = x) \geq 0 \; \forall x$$

for discrete random variables and

$$\int_x p(X = x) = 1$$
$$p(X = x) \geq 0 \; \forall x$$

for random variables.

Normal Distributions

The most common distribution we will encounter will be the normal distribution, sometimes called the Gaussian distribution, parameter-

ized by the mean μ and the variance σ^2.

$$p(x) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} = (2\pi\sigma^2)^{\frac{-1}{2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) = \mathcal{N}(x; \mu, \sigma^2)$$

In multidimensional cases, where x is a vector (and therefore μ is a vector $\mu \in \mathbb{R}^n$ and the variance Σ is a matrix $\Sigma \in \mathbb{R}^{n \times n}$), the following generalization of this formula is used.

$$p(x) = \det(2\pi\Sigma)^{\frac{-1}{2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) = \mathcal{N}(x; \mu, \Sigma)$$

Joint Distributions, Conditional Probabilities, and Bayes Theorem/Rule

The *joint distribution* models how likely two events are when measured as single combined event. For instance, what is the likelihood of flipping two coins and both coming up heads? These are independent of each other. But what about the joint probability of a bad student (as measured through a survey of all the student's prior teachers) getting a good grade (as measured using a particular assignment)? These are clearly not independent notions.

The joint distribution of $p(\mathcal{X} = x)$ and $p(\mathcal{Y} = y)$ is

$$p(x, y) = p(\mathcal{X} = x \text{ and } \mathcal{Y} = y).$$

Two events are considered *independent* if

$$p(x, y) = p(x)p(y).$$

The joint distribution is essentially the intersection of the two distributions. One can, of course, look at the union of the two distributions

$$p(\mathcal{X} = x \text{ or } \mathcal{Y} = y) = p(x) + p(y) - p(x, y).$$

Another important idea is that of the conditional probability, when two events are not independent of each other.

$$p(x|y) = p(\mathcal{X} = x|\mathcal{Y} = y) = \frac{p(x, y)}{p(y)}.$$

Note this implies that $p(y) \neq 0$. When $\mathcal{X}$ and $\mathcal{Y}$ are independent, $p(x|y) = p(x)$.

Lastly, there is what is called the *Theorem of Total Probability*, which states the following (somewhat obvious but nevertheless important) fact.

$$p(x) = \sum_y p(x|y)p(y) \text{ or } \int p(x|y)p(y)dy$$

Bayes Rule is both important and obvious. By the definition of conditional probability, we know that

$$p(x|y)p(y) = p(x, y)$$
$$p(y|x)p(x) = p(x, y)$$

so that

$$p(x|y)p(y) = p(y|x)p(x)$$

This last line, rearranged by dividing one side by $p(y)$ leads to Bayes Rule:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}.$$

In many textbooks, $\eta = \frac{1}{p(y)}$ is called a *normalization factor* for ease of notation. One of the shocking things about Bayes Rule is how bad we are at applying it and having intuition about it, as the next example illustrates.

Bayes Rule Example

This is a famous example due to Yudkowsky. Imagine you have a population U susceptible to cancer C and a test T for cancer. Given a positive test $T = 1$ for cancer, how concerned should an individual be that s/he has cancer (i.e., that $C = 1$)? To answer this, we need some information about cancer rates and test efficacy. Suppose for the age range considered, we know that

$$\begin{aligned} p(T = 1|C = 1) &= 0.75 \\ p(T = 1|C = 0) &= 0.10 \\ p(C = 1) &= 0.01. \end{aligned}$$

That is, the test capture correct positives three quarters of the time, has false positives one tenth of the time, and there is a baseline rate of one in one hundred people this age with cancer. Now, for a particular individual who receives a positive test of $T = 1$, what is that likelihood that $C = 1$?

We want to calculate this using Bayes Rule, so we need $P(T)$, which is given by the total probability

$$p(T = 1) = p(T = 1|C = 1)p(C = 1) + p(T = 1|C = 0)p(C = 0) = 0.75 \times 0.01 + 0.1 \times 0.99 = 0.1065.$$

Then we calculate $P(C = 1|T = 1)$ using Bayes Rule.

$$p(C = 1|T = 1) = \frac{p(T = 1|C = 1)p(C = 1)}{p(T = 1)} = \frac{0.75 \times 0.01}{0.1065} \approx 7\%.$$

Hopefully this gives you a sense of how unintuitive correct interpretation of probabilities can be.

A visualization of this is shown below.

Expectation and Entropy

The *expectation* of a random variable is its average/mean value weighted by the probabilities of each variable.

$$E[\mathcal{X}] = \sum_x xp(x) \text{ or } \int xp(x)dx$$

Note that expectations satisfy linearity with respect to addition. The *covariance* is the expected deviation from the expectation/mean.

$$\begin{aligned} Cov[\mathcal{X}] &= E\left[[[\mathcal{X} - E[\mathcal{X}]]^2\right] \\ &= E\left[\mathcal{X}^2 - \mathcal{X}E[\mathcal{X}] - E[\mathcal{X}]\mathcal{X} + E[\mathcal{X}]^2\right] \\ &= E[\mathcal{X}^2] - E[\mathcal{X}]^2 \end{aligned}$$

Lastly, *entropy* $h(x)$ is an attempt to quantify the information contained in a signal about a distribution, based on the following two axiomatic starting points.

1. First, that the information should add if two random variables are independent

$$h(x,y) = h(x) + h(y) \text{ when} p(x,y) = p(x)p(y)$$

2. Second, that $p(x,y) = p(x)p(y)$ implies that the log of $p(x,y)$ must be taken in order to get the additive property.

These requirements, really just observations, lead to the definition $h(x) = -\log(p(x))$, where using base 2 corresponds to the notion of the number of *bits*. To find the average amount of information required to describe a distribution, we take the expectation.

$$H(x) = E(h(x)) = \sum_x p(x)h(x) = \sum_x -p(x)\log(p(x))$$

Uncertainty as a Result of Noise versus Uncertainty as a Result of Structure

Uncertainty in robotics comes from two primary sources, noise and knowledge. The more familiar example is noise, where we might

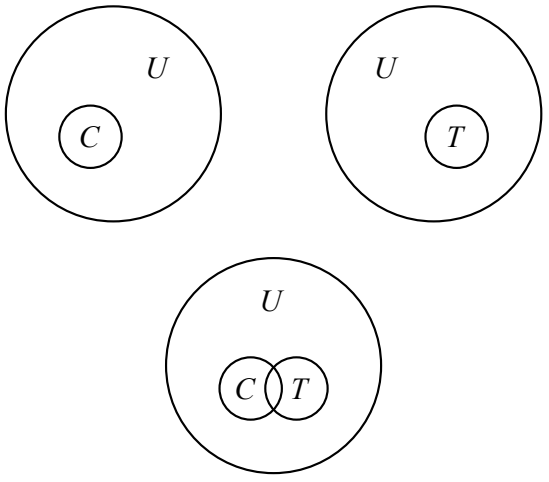


Figure 11: A visual representation of the probabilities of a population U being susceptible to cancer C and a test T for cancer. A positive test $T = 1$ does not always correspond to having cancer $C = 1$.

expect a range sensor to be able to detect the distance to the nearest object at which it points.

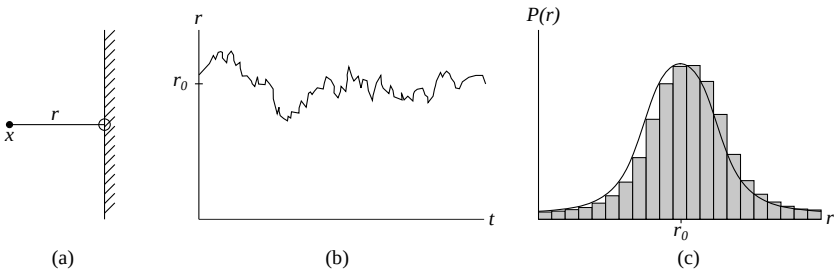


Figure 12: (a) The location x of a robot with a range sensor that detects a wall at a distance r. (b) Noisy measurements from the range sensor. (c) A numerical approximation showing the likelihood that the wall is actually distance r_0 away from x.

This range sensor will, of course, not give a static, perfect and perfectly constant signal. Instead, it will be close to the correct value, but with a contribution of uncertainty—often modeled as noise.

$$r(t) = r_{true}(t) + noiseterms = r_{true}(t) + dw \quad (dw \in \mathcal{N}(x; \mu, \sigma^2))$$

Another type of uncertainty, and one that is often more important in robotics, is that of *structure* uncertainty, possibly arising from the world and our knowledge of the world. For instance, imagine that a robot has a *door* sensor. (This might be a camera that rotates on top of a robot combined with a perception capability for detecting doors.) Let's say that a door can only be sensed if the robot is not overly oblique—a 45 degree angle is required relative to the side of the door in order to detect it. Then detecting a door implies that one is somewhere in a *map* defined by the sensor characteristics, as shown below.

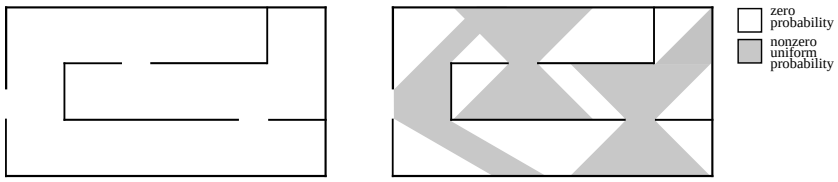


Figure 13: Left: An environment with doors and hallways. Right: The shaded regions indicate where a door can be detected using the measurement model described above.

Exercises and Questions

1. How does noise impact the map from the door function? (roughly, white → light grey, and black → dark grey)

2. What other variables could impact the probabilities? (e.g., angle of incidence uncertainty, obstruction/occlusion, role of orientation, speed of motion, distance-to-door)

3. How should one choose to move once in a region where there is a detected door?

4. How will the estimate of state evolve as a function of motion?

5. How would you describe the difference between uncertainty due to structure versus uncertainty due to noise, specifically in terms of entropy? (Imagine you are comparing GPS (possibly with lots of error) to the door sensor.)

Beliefs and Particle Filters

In this section we will always assume that our prior state and current control are sufficient to estimate the current state. That is,

$$p(x_t|x_{0:t-1}, z_{0:t-1}, u_{0:t}) = p(x_t|x_{t-1}, u_t).$$

where x_t is the state, z_t is the measurement, and u_t is the control. Moreover, we will similarly assume that the measurement only depends on the current state:

$$p(z_t|x_{0:t}, z_{0:t-1}, u_{0:t}) = p(z_t|x_t).$$

Transition Models

Probabilities *transition* from one time to another. We think of this as a dependence of x_t on the input and the prior state, leading to an observation.

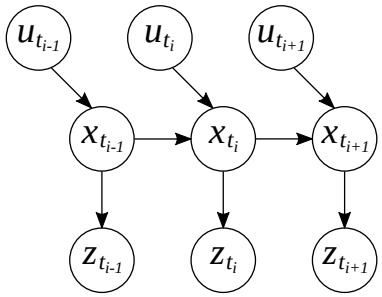


Figure 14: Controls $u_{t_{i-1}}$ affect states x_{t_i} which affect measurements z_{t_i}.

These models will typically come from differential equations like

$$\dot{x} = f(x, u) + noise$$

or

$$\dot{x} = f(x, u + noise).$$

(Think about what the difference is between these two situations!)

If a system is linear, then the transition function will be the state transition matrix we talked about earlier in class. That is:

$$\mathcal{X}_{t_{i+1}} = \Phi(t_i, t_{i+1})\mathcal{X}_{t_i}$$

where Φ satisfies the ordinary differential equation for the state transition matrix. If the ordinary differential equation is nonlinear, then one needs to discretize $f(x, u)$ over the time step. For example, one could use Euler integration

$$x_{t_{i+1}} = x_{t_{i+1}} + dtf(x_{t_{i+1}}, u_{t_{i+1}})$$

or a Runge-Kutta scheme.

Moreover, typically there will be uncertainty as mentioned above, and that uncertainty might enter the ordinary differential equation in terms of a completely separate vector field. Figuring out how to incorporate uncertainty systematically, even for additive noise models, can be very challenging for nonlinear systems because noise may make a system violate nonlinear constraints (e.g., a vehicle slipping sideways).

Example:

$$\dot{x} = u \quad x \in \mathbb{R}^2 \quad u_1 = 1 \quad u_2 = 1$$

The figure below shows the resulting state transition. The expected state changes if $\dot{x} = u + noise$. How would one need to renormalize the result?

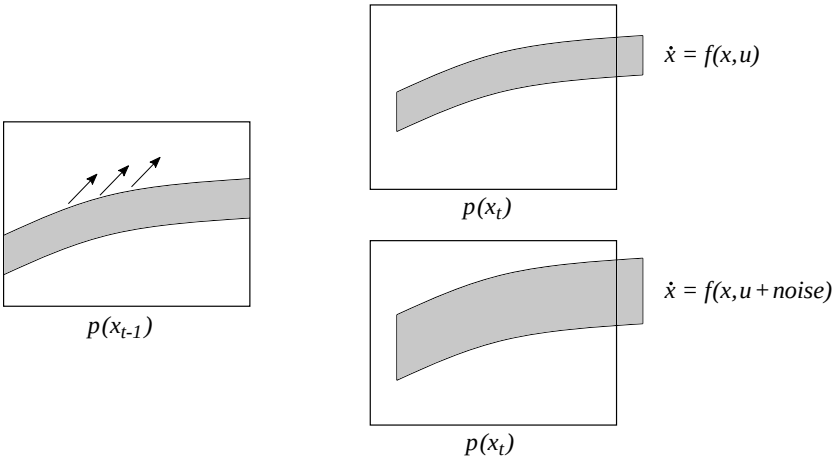


Figure 15: Left: The probability distribution at time $t-1$. We believe that x is somewhere in this region. Right: The belief at time t. We know the dynamics, so we predict that x has moved up and to the right, but the distribution becomes wider and more diffuse if there is noise in the control signal. The next step is to normalize the new belief.

Example: What if the dynamics are of the form

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

or

$$\dot{x} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u?$$

How would the evolution of distributions change?

Measurement Models

For any given sensor, a *measurement model* tells us how likely a given state is to have generated that measurement. We denote such a probability by $p(z_t|x_t)$. For instance, for the door sensor we have mentioned (assuming it needs to see both sides of the door), we get something like that shown below.

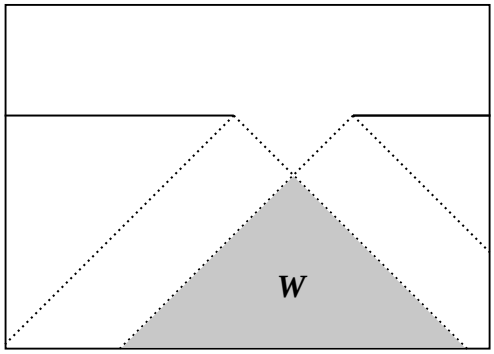


Figure 16: A measurement model for the door sensor. If the robot needs to see both sides of the door to detect it, it will observe a door only when it is in the region W.

If the area of the shaded region is W, then the measurement model $p(z_t = 1|x_t)$ is

$$p(z_t = 1|x_t) = \begin{cases} 1 & \text{if } x_t \in W \\ 0 & \text{else} \end{cases}$$

while its inverse is the following.

$$p(x_t|z_t = 1) = \begin{cases} \frac{1}{area(W)} & \text{if } x_t \in W \\ 0 & \text{else} \end{cases}$$

How would this model change if the door sensor were unreliable? That is, within the shaded region W the door sensor only correctly detects a door 80% of the time and outside of the region W it incorrectly detects a door 20% of the time?

Belief

To distinguish between distributions at different times t, we use the notion of a *belief* at time t to facilitate updating distributions.

$$bel(x_t) = p(x_t|z_{0:t}, u_{0:t})$$

and

$$\overline{bel}(x_t) = p(x_t|z_{0:t-1}, u_{0:t})$$

The second belief function acts as an intermediate step that *predicts* the belief based on a prior belief.

Bayes Filter

The general form of a Bayes filter has a prediction step and a reweighting step based on a measurement. Remember that the measurement is a single vector, not itself a distribution, but as soon as the measurement is known we can talk about how likely any given state is to have generated that measurement.

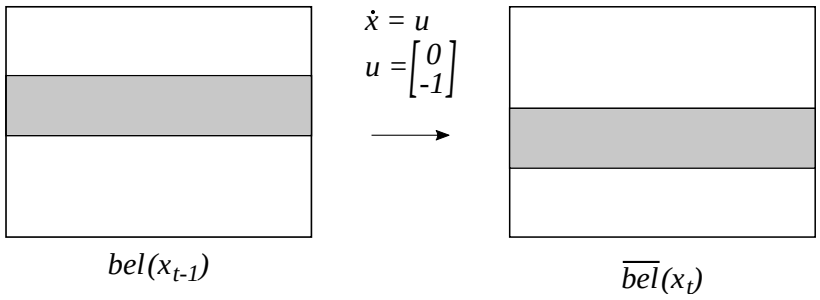


Figure 17: Left: The belief at time $t-1$. Right: The "belief bar" predicts the belief at time t based on the the belief at time $t-1$ and the control input, but not the measurement.

The first step in a Bayes filter is to update the belief with a prediction that generates a belief for *all possible* x_t:

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \; \forall x_t.$$

Note that this is a total probability calculation for each state x_t.

The second step in a Bayes filter is to update the belief with the measurement:

$$bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t) \; \forall x_t.$$

This is a conditional probability, conditioned on z_t. (Note that η is a normalization constant to keep the belief integrating to 1.) A geometric example of a belief update in both stages is below.

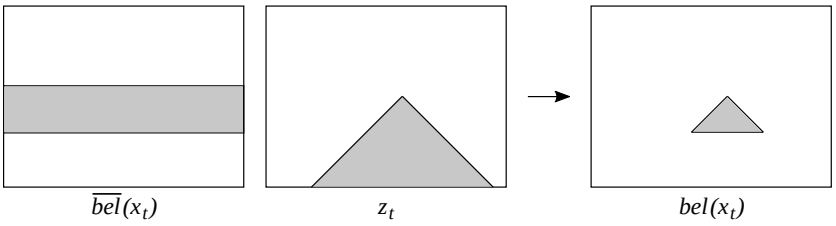


Figure 18: The $\overline{bel}(x_t)$ based on the previous belief $bel(x_{t-1})$ and the control input u_t and the measurement z_t are combined to result in a new distribution $bel(x_t)$.

Note that at the end the area of the distribution will not be the same as the area of either shaded region, so normalization is required, even in the case where probabilities are being generated purely from geometry-based reasoning.

Nonparametric Filtering: Particle Filters

Particle Filters are a way of using samples from a distribution to model the impact of nontrivial/nonlinear transition functions (typically arising from physics or some other principled modeling approach).

We start with a set of M samples.

$$\mathcal{X}_t = x_t^{[1]}, x_t^{[2]}, x_t^{[3]}, ..., x_t^{[M]}$$

These samples are sampling proportional to the distribution itself, as illustrated below (where the dashes are each sample, and are more dense in high probability regions). These samples are then mapped through some potentially nonlinear mapping $g(x)$ to obtain a sampling of a new distribution. This new distribution exists, but will generally never be known exactly; instead, it will be represented by the particles themselves.

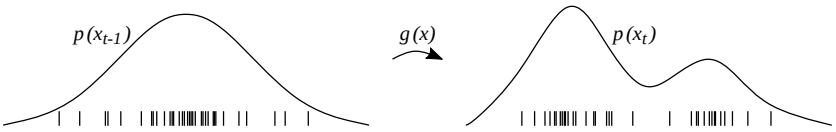


Figure 19: Particles (represented by small vertical lines) sampled from the distribution $p(x_{t-1})$ are transformed through mapping $g(x)$ to obtain a sampling of new distribution $p(x_t)$.

The steps of a particle filter follow the requirements of a Bayes Filter, using a prediction step and measurement step at each time. Note

Particle Filter Algorithm

$\mathcal{X}_t = \{\}$

$\overline{\mathcal{X}}_t = \{\}$

1: **for** $m = 1, ..., M$ **do**
2: sample $x_t^{[m]}$ from $p(x_t|u_t, x_{t-1}^{[m]})$
3: $w_t^{[m]} = p(z_t|x_t^{[m]})$
4: add the pair $(x_t^{[m]}, w_t^{[m]})$ to $\overline{\mathcal{X}}_t$
5: **end for**
6: **for** $m = 1, ..., M$ **do**
7: draw i with probability $w_t^{[m]}$
8: add $x_t^{[i]}$ to $\mathcal{X}_t$
9: **end for**

Return $\mathcal{X}_t$

that normalization is implicit because of the resampling procedure. An example of what this might look like is below.

Some notes on implementation are warranted here. First, the prediction step is executed in a couple of ways. What you are formally supposed to do is predict the distribution at time t assuming x_t as an initial condition, and then sample from the resulting distribution. This, of course, would be computationally prohibitive. Instead, we predict (typically using a differential equation) and include noise either by a) adding noise to the result and using the outcome as x_t, or b) adding noise to the control u and using the resulting state as x_t. (What we *do not* do, even if it would be more rigorous in many cases, is simulate many random executions from x_{t-1} to generate a full distribution at t and then sample from the result.)

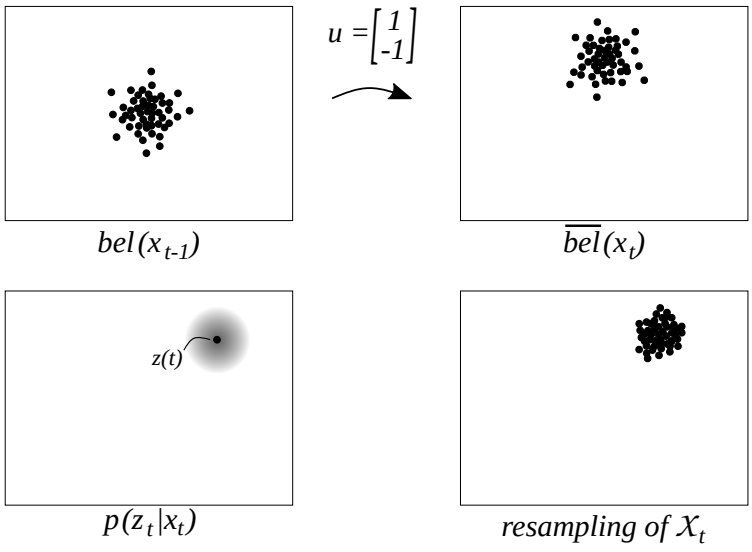
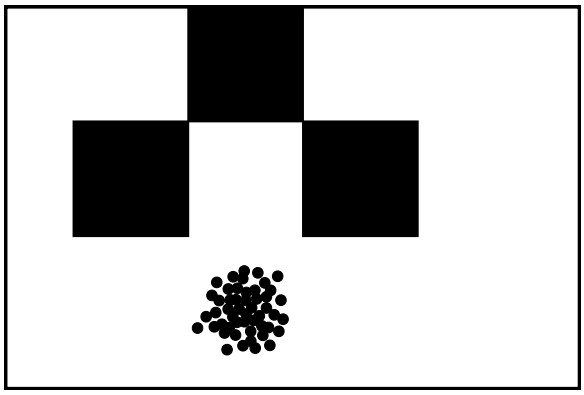


Figure 20: We start with the belief at time $t-1$, then predict $\overline{bel}(x_t)$ using the dynamics $\dot{x} = u$ and some amount of noise. Next we use our measurement z_t to calculate weights for the particles, and then resample from the weighted distribution.

Second, the resampling phase in line 7 is important, and hopefully intuitively makes sense. However, implementing it is not typically immediately obvious. To implement it, you can normalize the weights so that they all add up to one:

$$\tilde{w}_t^{[m]} = \frac{w_t^{[m]}}{\sum_i w_t^{[i]}}$$

and then randomly sample a number $rand[i]$ from the uniform distribution on $[0, 1]$. Then we find the i for which $\sum_{j=1}^{i} \leq rand[i] < \sum_{j=1}^{i+1}$ and add $x_t^{[m]}$ to $\mathcal{X}_t$.



Active Learning

Now we have our first opportunity to think about active learning strategies, where actions will have consequence for knowledge. For instance, in the image below, given the geometry of the environment and the distribution of particles, how should I choose u if $\dot{x} = u$ to best know x_t if all I have is a contact sensor?
Question: How would you generate a cost function that, through minimization as a function of u, achieves the desired result?

Kalman Filters

Our goal is to find a combination of two distributions of known covariance and combine them to minimize the covariance of the state estimate. We will approach this through one of several Kalman filter derivation strategies; ours focuses on the Kalman filter as the optimal *linear* filter (not necessarily focusing on normal distributions, though you can think of it in those terms if you like). As a result, we will use optimization to derive the filter.

Set up

We will make several assumptions. First, we assume a linear, discrete-time system.

$$\dot{x} = Ax + Bu$$

so that we have an associated discrete-time linear system.

$$\begin{aligned} x_k &= A_k x_{k-1} + B_k u_k + w_k \\ z_k &= C_k x_k + v_k \end{aligned}$$

The output is z_k and is impacted by measurement noise v_k of covariance R_k, which is uncorrelated from the state. The process noise w_k in the dynamics has covariance Q_k, also uncorrelated from the state. The input u_k is assumed known. The estimate of x_k will be denoted $\hat{x}_k$.

Prediction Step

Define

$$P_{k|k-1} = E[(x_k - \hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1})^T]$$

which is the covariance of the prediction step of the state error. Importantly, assume that the estimate dynamics are the same as the true dynamics (here we assume that the noise w_k is not correllated with the dynamics), so that

$$\hat{x}_{k|k-1} = A_k \hat{x}_{k-1|k-1} + B_k u_k.$$

This implies that

$$x_k - \hat{x}_{k|k-1} = A_k(x_k - \hat{x}_{k-1|k-1}) + w_k.$$

If we evaluate $P_{k|k-1}$, we get

$$\begin{aligned}
P_{k|k-1} &= E[(x_k - \hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1})^T] \\
&= E[(A_k(x_k - \hat{x}_{k-1|k-1}) + w_k)(A_k(x_k - \hat{x}_{k-1|k-1}) + w_k)^T] \\
&= E[(A_k(x_k - \hat{x}_{k-1|k-1}))(A_k(x_k - \hat{x}_{k-1|k-1}))^T] + E[w_k(A_k(x_k - \hat{x}_{k-1|k-1}))^T] \\
&\quad + E[(A_k(x_k - \hat{x}_{k-1|k-1}))w_k^T] + E[w_k w_k^T] \\
&= E[(A_k(x_k - \hat{x}_{k-1|k-1}))(A_k(x_k - \hat{x}_{k-1|k-1}))^T] + E[w_k w_k^T] \\
&\quad \text{because the noise is uncorrelated from the state} \\
&= A_k E[(x_{k-1} - \hat{x}_{k-1|k-1})(x_{k-1} - \hat{x}_{k-1|k-1}))^T] A_k^T + E[w_k w_k^T] \\
&= A_k P_{k-1|k-1} A_k^T + Q_k.
\end{aligned}$$

So, as a result we have a prediction step of the following form.

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_k$$

Measurement Update

Now let us consider the measurement update. We will assume that the state update is a linear combination of the prediction update $x_{k|k-1}$ and the measurement z_k. That is, we assume it is of the following form.

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - C_k \hat{x}_{k|k-1})$$

Here we will take the derivative of a real-valued function of $P_{k|k}$ with respect to K_k and set it equal to zero to find the optimum choice of K_k.

First, look at $P_{k|k}$.

$$\begin{aligned}
P_{k|k} &= E[(x_k - \hat{x}_{k|k})(x_k - \hat{x}_{k|k})^T] \\
&= E[(x_k - (\hat{x}_{k|k-1} + K_k(z_k - C_k \hat{x}_{k|k-1})))(x_k - (\hat{x}_{k|k-1} + K_k(z_k - C_k \hat{x}_{k|k-1})))^T] \\
&= E[((x_k - \hat{x}_{k|k-1}) - K_k((\underbrace{C_k x_k + v_k}_{z_k}) - C_k \hat{x}_{k|k-1}))((x_k - \hat{x}_{k|k-1}) - K_k((\underbrace{C_k x_k + v_k}_{z_k}) - C_k \hat{x}_{k|k-1}))^T] \\
&= E[((x_k - \hat{x}_{k|k-1}) - K_k(C_k(x_k - \hat{x}_{k|k-1}) + v_k))((x_k - \hat{x}_{k|k-1}) - K_k(C_k(x_k - \hat{x}_{k|k-1}) + v_k))^T] \\
&= P_{k|k-1} - K_k C_k P_{k|k-1} - P_{k|k-1} C_k^T K_k^T + K_k(C_k P_{k|k-1} C_k^T + R_k) K_k^T.
\end{aligned}$$

(Note that the v_k drop out because they are uncorrelated to the state.)

This is where the big idea in Kalman 1960 is. Kalman realized that the manifold of all possible measurements is a linear manifold, generated through convolution of the random signal and the dynamics. As a result, a major theorem from Hilbert implies that the optimal solution *must* be a linear projection. So, although we are choosing to only consider linear filters, Kalman showed that in a relatively broad set of situations linear filters are in fact the only filters that need to be considered.

Now we optimize something with respect to K_k. It turns out that the *trace* of $P_{k|k}$ is a good choice because a) it is equal to the sum of the errors squared in the state and b) because it leads to nice statements of the derivative. One could, of course, use other measures (e.g., the determinant of $P_{k|k}$), but those do not lead to nice derivatives.

The trace $tr(A)$ has two properties we will need:

$$\frac{\partial}{\partial X} tr(XA) = A^T$$
$$\frac{\partial}{\partial X} tr(XAX^T) = 2XA$$

Both of these can be confirmed by direct calculation.

Now we take the derivative and set it equal to zero.

$$\begin{aligned}
\frac{\partial}{\partial K_k} P_{k|k} &= \frac{\partial}{\partial X} tr\left(P_{k|k-1} - K_k C_k P_{k|k-1} - P_{k|k-1} C_k^T K_k^T + K_k (C_k P_{k|k-1} C_k^T + R_k) K_k^T\right) \\
&= \frac{\partial}{\partial X}\left(tr(P_{k|k-1}) - tr(K_k C_k P_{k|k-1}) - tr(P_{k|k-1} C_k^T K_k^T) + tr(K_k (C_k P_{k|k-1} C_k^T + R_k) K_k^T)\right) \\
&= -2(C_k P_{k|k-1})^T + 2K_k (C_k P_{k|k-1} C_k^T + R_k) = 0 \\
&\implies K_k = P_{k|k-1} C_k^T [C_k P_{k|k-1} C_k^T + R_k]^{-1} \\
&\implies P_{k|k} = P_{k|k-1} - K_k C_k P_{k|k-1}
\end{aligned}$$

Notion of Optimality

What does "the optimal linear filter" mean in this setting? It should mean that if you look at nearby linear filters, K_k is better than all of them in some way. Specifically, if you do Monte Carlo simulations of the process and measurement noise, you should see that across all of them the optimal choice of K_k leads to the best performance on average.

Brief note on Kalman Smoothers

We have made a big point about state statistics being sufficient. But is it possible to generate a linear filter that is better than Kalman filter? The answer is yes, but only if we take advantage of measurement data in the future to update estimates in the past. This is called smoothing, rather than filtering, and leads to even better estimates.

Entropy

Recall that the goal of information is fourfold:

1. the information about something should increase monotonically as measurement are obtained
2. $\mathcal{I}(p) \geq 0$
3. $\mathcal{I}(1) = 0$
4. Independent events ($\mathcal{I}(p_1 p_2) = \mathcal{I}(p_1) + \mathcal{I}(p_2)$) should have information that sums.

Shannon was interested in how many questions of a yes/no type will resolve the state on average. He noticed that log functions meet all these requirements of information, and that in fact they are the only functions that do so. This leads to the expressions:

$$h = \log(\frac{1}{p(x)}) = -\log p(x) \quad H = \sum_i -p(x_i) \log p(x_i)$$

where H is the *entropy* measuring information. In base 2, H is measured in *bits*.

Coin flip example: ($H = heads$, $T = tails$.) Given a fair coin, $p(H) = p(T) = 0.5$. leads to $H = 2$ *bits*, as expected. However, given an unfair coin $p(H) = 0.899972$, we get $H = 1$ *bit*; that is, there is only one bit of information, on average, in every two coin flips.

As a consquence, the number of bits suggests that we should be able to ask better questions. For instance, in two coin flips, we have the option of asking the query HH, which has roughly an 80% likelihood of being true (thus terminating our need to ask questions). If the answer is no, then we could HT, then followed by the question TH if needed. On average, this leads to a scheme that requires 1.3 questions per two coin flips. Shannon's 1 bit entropy is a bound, and as the number of flips goes up our ability to approximate the bound will become better.

Differential Entropy

Entropy can be generalized to continuous variables using

$$H = -\int_X p(x) \log(p(x)) dx$$

which seems like a perfectly reasonable generalization. However, if one approximates this integral with a summation, investigation of the summation shows that the $\log(0)$ shows up, making the differential entropy appear to be infinite. Moreover, this definition can lead to negative entropy.

Example entropy for several distributions:

1. Normal distribution: $\log(\sigma\sqrt{2\pi e})$
2. Uniform distribution $f = \frac{1}{b-a}$ has an entropy: $\log(b-a)$
3. Multivariate Normal distribution: $\log((2\pi e)^N \det(\Sigma))$

Relative Entropy

The *relative entropy* fixes many of the problems with differential entropy by comparing information in two distributions. It is defined as the following.

$$h(P||Q) = \log\left(\frac{P}{Q}\right) \quad H = -\sum_i P(x_i)\log\left(\frac{P(x_i)}{Q(x_i)}\right)$$

This definition indicates how well P captures information in Q. (See link about space worms.) The relative entropy also generalizes to continuous variables well.

$$H = -\int_X P(x)\log\left(\frac{P(x)}{Q(x)}\right)dx$$

Maximum Likelihood & Fisher Information

Reminder: Let X be a random variable and $Y = g(X)$. Then Y is also a random variable since for a particular outcome $X = x$, $Y = g(X = x)$. Derivatives of Y are also random variables, so we can evaluate expectations and variances of a number of functions of random variables.

Likelihood

Suppose we have a model with unknown parameter or set of parameters θ. We have observations of a random variable X with probability $f(x|\theta)$ depending on a fixed parameter θ. For a given observation $X = x$, we can define the likelihood of a value of θ as

$$\mathcal{L}(\theta|x) = f(x|\theta).$$

It's important to note that *the likelihood is not a probability* and does not have the same properties as a probability density function or probability mass function. Specifically, the integral over possible values of θ does not need to equal one. If the likelihood function were a pdf, we might take the expected value of θ as the estimate. Since the likelihood is not a pdf and we have no prior about our parameter θ the best estimate we can choose will be based on the maximum of the likelihood function.

Coin Flip Example. Take the example of a coin flip with unknown fairness, $\theta = p(heads)$. We choose our random variable to be a pair of IID flips. If our measurement is $X = HH$, and assume the coin is fair ($\theta = 0.5$), then the probability of this event is

$$f(X = HH|\theta = 0.5) = 0.5^2 = 0.25.$$

The likelihood function for this observation is then $\mathcal{L}(\theta|X = HH) = \theta^2$.

Based on this measurement and no prior information, our best estimate of θ is

$$\hat{\theta} = \underset{\theta \in [0,1]}{\arg\max}\ \mathcal{L}(\theta|x) = 1.$$

This estimate changes when we get a different outcome of 2 coin flips or when we use a different random variable. One should expect that the estimate should get better when our random variable contains more information, regardless of the observed outcome of that random variable. For the coin, we can choose a new random variable to be a sequence of 3 coin flips. When we do this we also see that the peak of our likelihood functions becomes sharper.

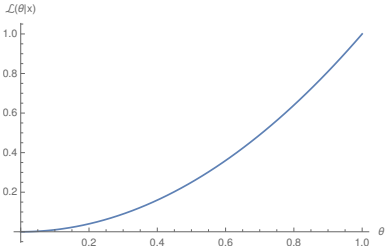


Figure 21: **The likelihood function of the fairness of a coin given two coin flips resulting in heads.** The maximum likelihood estimate $\hat{\theta}$ is 1. This illustrates a key difference between the likelihood and a probability function. Notice that the integral over the possible values of θ is much less than one. The integral is actually $1/3$.

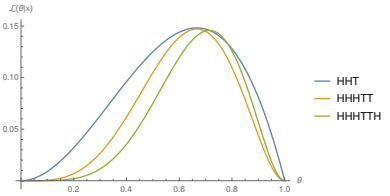


Figure 22: Example likelihood functions of 3 different random variables.

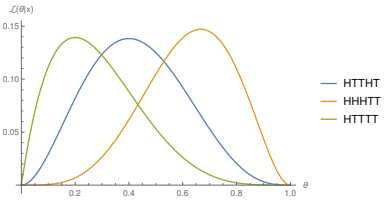


Figure 23: Example likelihood functions of 3 different observations of the same random variable. The information content of the random variable increases with the number of coin flips. With increasing information from the random variable, the spread of the likelihood over θ decreases, suggesting a decrease in variance. Note that the values of the curves have been shamelessly scaled to make this figure more readable.

Maximum Likelihood Estimation

Given the curves shown in Fig. **??**, how should we choose an estimate of θ? We do what we always do to find extrema; take the derivative and set it equal to zero. However, the likelihood function can take some pretty inconvenient forms. When our random variable, X, is actually a set of random variables $\{x_1, x_2, ..., x_n\}$ the likelihood usually has the form of a product of probabilities. For a set of independent random variables like the coin flips,

$$\mathcal{L}(\theta|X = \{x_1, x_2, ..., x_n\}) = \prod_1^n f(x_i|\theta)$$

Rather than iteratively apply the product rule, it is common to use the natural log of the likelihood. Since the logarithm is a strictly increasing function, maximizing the log of the likelihood also maximizes the likelihood. Conveniently, multiplication in log space is equivalent to addition, so we get the log-likelihood

$$l(\theta|x) = log\mathcal{L}(\theta|X = \{x_1, x_2, ..., x_n\}) = \sum_1^n log(f(x_i|\theta)),$$

so we can simply sum the derivatives and avoid product rule. Note that we did not choose the natural logarithm because we are trying to measure some sort of information, so it is just a coincidence that it shows up here as well as in the definition of the Shannon entropy and relative entropy. Generally, the derivative of the log-likelihood is

$$\frac{\partial}{\partial\theta} log(f(x|\theta)) = \frac{D_\theta f(x|\theta)}{f(x|\theta)}.$$

To find the maximum likelihood estimate, simply solve for θ in $D_\theta f(x|\theta)/f(x|\theta) = 0$

Fisher Information

Up until now, we have assumed that we already have all of our observations, but when trying to estimate a parameter of a robot, we have control authority over the random variable we are observing. If we get to choose our random variable, we could use two strategies. First, we could insist on using a lot of measurements of the same thing as our random variable and hope that, as with the coin flipping example, the spread of theta decreases with increasing flips. Alternatively, we can selectively observe random variables with high information. To do this, we need to come up with a way to *measure the information about θ contained in X*.

For an initial guess of θ and an observation $X = x$, $D_\theta l(\theta|x)$ close to zero implies that the observation was expected, so it doesn't provide much new information. On the other hand, a low probability

event with $|D_\theta l(\theta|x)| >> 0$ implies that X contains a lot of information about θ.

Thus, we can use $[D_\theta l(\theta|x)]^2$ to measure the information provided by X.

Because X is a random variable, we take the expectation over x to be the Fisher information.

$$I(\theta) = E[D_\theta l(\theta|x)^2] = \int [D_\theta l(\theta|x)]^2 f(x|\theta) dx = \int \frac{D_\theta f(x|\theta)^2}{f(x|\theta)} dx$$

There are a couple of other ways we can write the Fisher information by taking advantage of the properties of the probability $f(x|\theta)$. First, we know that the integral of $f(x|\theta)$ over x must be equal to 1, so if f is differentiable we know that,

$$\int D_\theta f(x|\theta) dx = D_\theta \int f(x|\theta) dx = 0.$$

Furthermore, when f is twice differentiable,

$$\int \frac{\partial^2}{\partial \theta^2} f(x|\theta) dx = D_\theta^2 \int f(x|\theta) dx = 0$$

Using these properties of f, we get that the expected value of the derivative of log-likelihood is 0.

$$E[D_\theta l(\theta|x)] = \int D_\theta l(\theta|x) f(x|\theta) dx = \int \frac{D_\theta f(x|\theta)}{f(x|\theta)} f(x|\theta) dx = \int D_\theta f(x|\theta) dx = 0$$

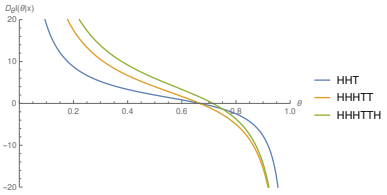


Figure 24: Derivative of the log-likelihood of 3 different random variables.

Now let's look at a plot of the derivative of the log-likelihood (Fig. **??**). When there are a lot of values of θ that put the derivative near zero this suggests there is not a lot of information about θ contained in the random variable X because we are less certain about the particular value of θ where there is a maximum. When the slope of $D_\theta l(\theta|x)$ is steeper it is clear that the value of θ that is equal to zero is more unique. Another way to say this is that when the variance of $D_\theta l(\theta|x)$ is high, we are more certain of our estimate $\hat{\theta}$.

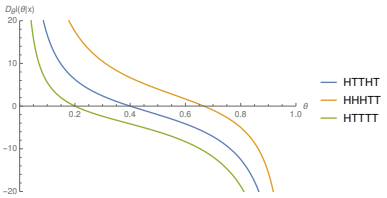


Figure 25: Derivative of the log-likelihood of 3 different measurements of the same random variable. When there are a lot of values of θ that are near zero this suggests there is not a lot of information about θ contained in the random variable.

When we consider the value of the log-likelihood derivative for a particular θ to be a random variable that occurs with probability $f(x|\theta)$, we can view the Fisher information as a variance,

$$I(\theta) = Var[D_\theta l(\theta|x)].$$

Often the most convenient choice for calculating the Fisher information uses the second derivative of the log-likelihood.

$$D_\theta^2 l(\theta|x) = \frac{\partial}{\partial \theta}\left[\frac{D_\theta f(x|\theta)}{f(x|\theta)}\right] = \frac{D_\theta^2 f(x|\theta) - D_\theta f(x|\theta)^2}{f(x|\theta)^2} = \frac{D_\theta^2 f(x|\theta)}{f(x|\theta)} - [D_\theta l(\theta|x)]^2$$

$$E[D_\theta^2 l(\theta|x)] = \int \left[\frac{D_\theta^2 f(x|\theta)}{f(x|\theta)} - [D_\theta l(\theta|x)]^2\right] f(x|\theta) dx = \int D_\theta^2 f(x|\theta) dx - \int [D_\theta l(\theta|x)]^2 f(x|\theta) dx$$

Finally, we can write another formula for the Fisher information,

$$I(\theta) = -E[D_\theta^2 l(\theta|x)] = -\int D_\theta^2 log(f(x|\theta)) f(x|\theta) dx.$$

This also extends to an $N \times 1$ vector of parameters $\theta = [\theta_1, \theta_2, ..., \theta_N]^T$. In this case the Fisher information becomes an $N \times N$ matrix with each entry being given by,

$$[\mathcal{I}(\theta)]_{ij} = -E\left[\frac{\partial^2}{\partial\theta_i \partial\theta_j} log f(x|\theta)\right].$$

So what exactly is the Fisher information measuring? It is measuring the *information about θ contained in X.* This also gives us a measure of how sure we are about our maximum likelihood estimate. If the Fisher information is low—like our 2 coin flip example—we are less sure of our estimate than if we had calculated a likelihood from a random variable containing the outcomes of 5 flips.

Connection to Kullback-Leibler Divergence

The Fisher information is also related to the relative entropy or Kullback-Leibler divergence of two parameterized distributions describing a random variable X. Say we have two distribution $f(x|\theta)$ and $f(x|\theta_\epsilon)$, recall that there Kullback-Leibler divergence is given by,

$$D_{KL}[f(x|\theta)||f(x|\theta_\epsilon)] = \int f(x|\theta) log\left(\frac{f(x|\theta)}{f(x|\theta_\epsilon)}\right) dx.$$

If we take the Taylor series approximation of this about $\theta = \theta_0$, where $\theta_\epsilon = \theta_0 + \epsilon$, we get,

$$D_{KL}[f(x|\theta_0)||f(x|\theta_\epsilon)] = D_{KL}[f(x|\theta_0)||f(x|\theta_\epsilon)] + \epsilon \left.\frac{\partial}{\partial\theta} D_{KL}[f(x|\theta)||f(x|\theta_\epsilon)]\right|_{\theta=\theta_0} + \frac{\epsilon^2}{2} \left.\frac{\partial^2}{\partial\theta^2} D_{KL}[f(x|\theta)||f(x|\theta_\epsilon)]\right|_{\theta=\theta_0} + \mathcal{O}^3$$

If we take the limit as $\epsilon \to 0$, the first two terms are zero, because the Kullback-Leibler divergence has an absolute minimum of 0 when $f(x|\theta) = f(x|\theta_\epsilon)$, so its derivative at θ_0 is 0. This leaves only the second-derivative term.

$$\frac{\partial}{\partial\theta} D_{KL} = \int D_\theta f(x|\theta) log\left(\frac{f(x|\theta)}{f(x|\theta_\epsilon)}\right) + D_\theta f(x|\theta) dx$$

$$\frac{\partial^2}{\partial\theta^2} D_{KL} = \int D_\theta^2 f(x|\theta) log\left(\frac{f(x|\theta)}{f(x|\theta_\epsilon)}\right) + \frac{D_\theta f(x|\theta)^2}{f(x|\theta)} + D_\theta^2 f(x|\theta) dx$$

The first term goes to zero when $\epsilon = 0$ and the last term is zero because $f(x|\theta)$ is a probability over x. This leaves

$$\frac{\partial^2}{\partial\theta^2} D_{KL} = \int \frac{D_\theta f(x|\theta)^2}{f(x|\theta)} dx = I(\theta),$$

so the second derivative of the K-L divergence for distributions parameterized by nearby values of θ is the Fisher information.

Calculating the Fisher Information & Choosing a Random Variable

Going back to our coin flip example, where we reasonably asserted that increasing the number of observed flips should increase our information about θ, we want to confirm that this assumption was correct. To do this, we calculate the Fisher information of 3 different random variables (2 flips, 3 flips, and 5 flips) and plot them across all possible values of θ.

Example of the Fisher Information Calculation for 2 coin flips.

The Fisher information is an expectation over all possible values of X, so let's first write down the possible values a pair of coin flips could take

$$X = [TT, HT, TH, HH],$$

and the likelihood of each of those outcomes is

$$\mathcal{L}_x(\theta) = \begin{cases} (1-\theta)^2 & x = TT \\ (1-\theta)\theta & x = TH \\ \theta(1-\theta) & x = HT \\ \theta^2 & x = HH \end{cases}$$

The log-likelihood is then

$$l_x(\theta) = \begin{cases} 2Log(1-\theta) & x = TT \\ Log(1-\theta) + Log\,\theta & x = TH \\ Log\,\theta + Log(1-\theta) & x = HT \\ 2Log\,\theta & x = HH \end{cases}$$

Taking the second derivative of that, we get

$$D^2_\theta l_x(\theta) = \begin{cases} -\frac{2}{(1-\theta)^2} & x = TT \\ -\frac{1}{(1-\theta)^2} - \frac{1}{\theta^2} & x = TH \\ -\frac{1}{\theta^2} - \frac{1}{(1-\theta)^2} & x = HT \\ -\frac{2}{\theta^2} & x = HH \end{cases}$$

We calculate the Fisher information by taking the expectation of this derivative over all possible x,

$$I(\theta) = \sum_X D^2_\theta l_x(\theta) * \mathcal{L}_x(\theta) = \frac{4}{(1-\theta)^2} + \frac{2}{\theta^2}.$$

Performing a similar calculation for random variables of 3 flips and 5 flips, we can see that our assumption about the relative amount of information about θ contained in each possible random variable was correct (fig. **??**). The Fisher information does increase with increasing number of coin flips.

Now, coin flips are a great example to help us reason about the information contained in a random variable, but they don't have a lot to do with robots. When trying to estimate a parameter of your robot or its environment, we also get to choose our random variable. For instance, you have the option to choose between two different sensor measurements. If one sensor measures 3 coin flips, the other sensor measures 5 coin flips, and there is no energetic cost between the two, you would choose the 5 coin flip sensor since it will on average contain more information about your unknown parameter.

Let's assume our random variable must be a set of measurements over a fixed time. What does the possible set of random variables that we can choose look like? Each element of the set is the measurements over a trajectory. We might want to estimate the mass of simple point-mass governed by $F = ma$ where we get to choose the forces applied over time and measure the acceleration, or we could estimate the length of a cart-pendulum by choosing the trajectory of the cart and measuring the angle. **Every choice of input $u(t)$ corresponds to a different choice of our random variable X.** In the case of the point mass or the cart-pendulum, one choice is to apply zero force or zero velocity for all time. This particular choice won't give us much information even if there is process noise and measurement noise that changes our measurements over time. However, if we choose a constant force or constant velocity, we can probably get a little more information. Finally, the best choice of a random variable is probably one where our inputs vary over time. We can find the optimum of this using our dynamics and measurement model to formulated an optimization on the Fisher information.

In order to optimize with respect to Fisher Information, we must choose a measure on the Fisher Information matrix. A few reasonable measures are to use the determinant, trace, or maximum/minimum eigenvalue of the matrix.

2 flips $\mathcal{I}(\theta) = \frac{2}{\theta - \theta^2}$
3 flips $\mathcal{I}(\theta) = \frac{3}{\theta - \theta^2}$
5 flips $\mathcal{I}(\theta) = \frac{5}{\theta - \theta^2}$

Figure 26: ÃćÂĂÂć

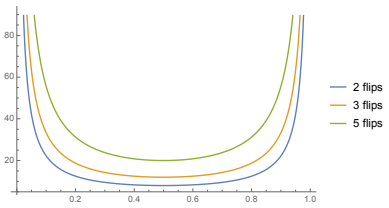


Figure 27: ÃćÂĂÂć **fix this!** The Fisher Information for Random Variables in the Coin Flip Example.

Posterior Probabilities & Infotaxis

Posterior Probability

Last lecture, we talked about how we could choose our random variable X to contain the most information about an unknown parameter θ regardless of the particular observation of X that we might get. To motivate this we used the example of a coin flipping experiment, where increasing the number of coin flips contained in our random variable increased the information we would get for any observation of the set of coin flips. By taking a specific observation we could use maximum likelihood estimation to estimate our model parameter θ.

However, that model of parameter estimation doesn't allow us to combine information we get from an observation with prior knowledge that we have about the model. This prior knowledge usually comes from some type of measurement, but might be based on assumptions of the probability of our parameter taking on certain values. Since we are often taking measurements sequentially in time, we can use our knowledge of the previous measurement to improve our estimate of the parameter value θ.

For the coin flip example from last lecture, we assumed we had no idea what the fairness of the coin might be. Now let's assume instead that we have observed a lot of these coins which are manufactured the same way and on average the coins are fair ($E[\theta] = E[p(heads)] = 0.5$) and are normally distributed with a variance of 0.01. This distribution defines our *prior* as

$$p(\theta) = \frac{1}{\sqrt{2\pi\sigma^2}}\, exp\left(-\frac{\theta - 0.5}{2\sigma^2}\right).$$

If we pull a coin from this distribution and flip it twice observing $x = HH$, what is the probability density function of θ of this particular coin? Bayes rule says that,

$$p(\theta|x) = \frac{f(x|\theta)p(\theta)}{p(x)}.$$

This is called the *posterior* probability distribution of θ given $X = x$. The posterior distribution is proportional to the $Prior \times Likelihood$, normalized by the total probability of x which in this case is given by $p(x) = \int_0^1 f(x|\theta)p(\theta)d\theta$.

This is the same concept behind the Bayesian filters discussed in lecture ***. In the case of the particle filter, our prior was the previous belief forward simulated based on the dynamics of the system. You'll recall that we reweighted those particles according to how likely they were given a new measurement. The new weights defined our posterior distribution and provided us with an improved estimate of

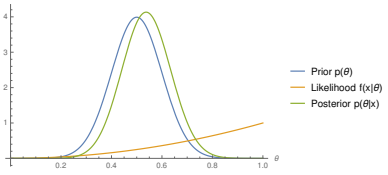


Figure 28: The Posterior Probability of the fairness of a coin.

the true state of the system. The same is true for our update on $p(\theta)$ in fig. 29.

Choosing Inputs Based on the Posterior

If our goal is to determine a parameter θ of our model, we discussed that we can choose the random variable that we observe based on our inputs. Choosing the optimal inputs involves maximizing a measure on information gain, such as the Fisher information or minimizing the entropy of your posterior. The general form our posterior will take is

$$p_x(\theta_0) = \frac{\mathcal{L}_{\theta_0}(x)p(\theta_0)}{\int \mathcal{L}_\theta(x)p(\theta)d\theta},$$

where x is our current measurement or a statistic on our history of measurements. An example of entropy minimization is a search process where the unknown location of an object, a door, or the source of a chemical plume is the unknown parameter. Search problems are often view as a balance between exploitation (going to the highest probability area of the posterior) and exploration (collecting more information to incorporate into the posterior).

Algorithm 2: Infotaxis - Entropy Reducing Control for Source/Object Localization

Initialize the entropy of the posterior distribution S and the prior $p(\theta)$.

while $S \neq 0$ **do**
  Take a measurement x
  Update the posterior $p_x(\theta)$
  Recalculate S from the posterior
  Calculate a control input $u_i = \arg\min_u E[\Delta S(u)]$
  Update the prior $p(\theta)$ based on last measurement
  Apply the input u_i
end while

Infotaxis

An example of an entropy reducing controller for chemical source location is infotaxis[8] The chemical source emits particles which the agent detects with a probability dependent on the distance from the source. The times and coordinates of previous hits along the trajectory $r(t)$ are stored in the random variable $\mathcal{T}_t$, which acts as the information signal. The posterior probability of the source location r_0 is then

$$P_t(r_0) = \frac{\mathcal{L}_{r_0}(\mathcal{T}_t)}{\int \mathcal{L}_{\mathbf{x}}(\mathcal{T}_t)d\mathbf{x}}.$$

[8] Vergassola, Villermaux, & Shraiman, *'Infotaxis' as a strategy for searching without gradients*, Nature, 2007.

where the posterior probability of all previously visited locations must be zero, since we know that we did not find the source in those locations. In other words, the total probability $P(r_0)$ is 0 for all previously visited locations and all other locations have equal probability. As a result, the total probability can be taken outside the integral which is only over unvisited points in the space.

Given this posterior probability, there are a few options about how to optimize your movements to find the source location. First, you can choose to use some sort of random walk around the space with not particular preference for what direction your next action will take you. Alternatively, you can move towards the area of highest probability. If that happens to be the source location, we're done. The new posterior is a delta function and the entropy goes to zero. However, if that's not the case we can update our posterior and move to the new highest probability area. This iterative process represents complete exploitation at every time step. We can also choose to move based on maximum information gained or maximum entropy reduction.

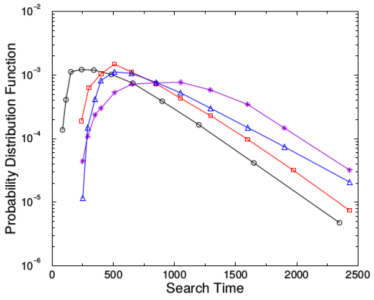


Figure 29: Results of a Monte Carlo test of the time to localization for different cost functions. **Blue:** The searcher moves to the neighbor with highest estimated probability according to $P_t(r_j)$. **Red:** Local maximization of $E[\Delta S(r_j)|r_j \neq r_0]$. **Black:** Infotaxis search strategy. **Purple:** Local maximization of the estimated number of hits $h(r_j)$.

Unfortunately, the entropy reduction is a function of the set of encounters/event $\mathcal{T}_t$ that might occur after applying the next input, making ΔS itself a random variable. Rather than directly maximizing our entropy reduction we must evaluate our possible actions based on the *expected* change in entropy. In a grid world, we can move to the 4 nearest neighbors or stay in place. If our current entropy is S, then for each of the five possible actions we could take, we will reduce our entropy to zero ($\Delta S(r_j) = -S$) with probability $P_t(r_j)$ if we encounter the source at the next step. All other possible entropy reductions with happen with probability $1 - P_t(r_j)$.

$$E[\Delta S(r_j)] = E[S_{P_{t+1}(r_0)} - S_{P_t(r_0)}] = P_t(r_j)(-S) + (1 - P_t(r_j))E[\Delta S(r_j)|r_j \neq r_0]$$

Within this expectation, there is another expectation we must consider, $E[\Delta S(r_j)|r_j \neq r_0]$. This expectation is the weighted sum of the entropy reduction for any possible number of hits k in the next Δt seconds,

$$E[\Delta S(r_j)|r_j \neq r_0] = \sum_k \rho_k(r_j)\Delta S_k.$$

In the case of the chemical plume in the infotaxis paper, they assume the number of hits is Poisson distributed, so $\rho_k(r_j) = \frac{h(r_j)e^{-h(rj)}}{k!}$. Rather than computing the sum over all possible k they make an approximation using only a small value of k.

Note that because we are limiting ourselves to only 5 possible input values, this is a local optimization method rather than a global method. This means that if there is more than one source, object or door, the agent will find only one and stop looking for any other sources once it finds one.

Discussion

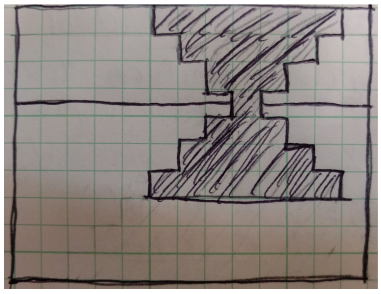


Figure 30: Infotaxis setup of door sensor

Initialize the prior as a uniform distribution and initialize entropy as the entropy of the prior.

while $S \neq 0$ **do**

Take a measurement x of 0 or 1

Update the posterior $p_x(\theta) = \frac{\mathcal{L}_{r_0}(x)p(r_0)}{p(x)}$

- $\mathcal{L}_{r_0}(x) =$ a uniform distribution over the 4 nearest squares in every direction.
- $p(r_0) =$
$$\begin{cases} 0 & for\ all\ visited\ squares \\ 1/(Total\ number\ of\ unvisited\ squares) & else \end{cases}$$
- $p(x = 1) = \frac{Number\ of\ black\ squares}{Total\ number\ of\ squares}$
- $p(x = 0) = 1 - p(x = 1)$

Recalculate S from the posterior

Calculate a control input $u_i = \arg\min_u E[\Delta S(u)]$

Update the prior $p(\theta)$ based on last measurement

Apply the input u_i

end while

Algorithm 3: Infotaxis for the door sensor

Process of infotaxis with the door sensor.

1. Initialize the belief over where the door sensor is as a uniform distribution over the whole search space. You will need to represent the belief as a 2D grid, each cell of the grid will be assigned a value representing the likelihood that the door is in this grid. The belief will be over the whole search space throughout the whole process.

2. Compute the entropy of the belief grid.

3. Take a binary measurement at the current location of the robot. You will need to simulate this measurement using the given measurement. Simulation of the measurement will be similar to flipping a potentially unfair coin.

4. Perform Bayesian update for the belief. The belief, as a 2D grid, is a discretized representation of the distribution $p(x)$, this will the prior belief. The given measurement model, denoted as $m(x, s, z)$, serves as the likelihood function. It describes the likelihood of taking the measurement z (which is either 0 or 1) given the robot position at x and the door location at s. In each time step, you know where the robot position is and what the measurement is, it is the door location s you are trying to estimate with the belief, so think about how to compute Bayesian posterior in this case. Remember to normalize your 2D grid after the Bayesian update.

5. Compute the entropy of the updated belief.

6. Now it is time to choose the next action. For each of the five candidate actions (left, right, up, down, stay), do the following:

   (a) Take the candidate action (hypothetically).

   (b) Take a hypothetical measurement of 1, which means you do detect the door after taking the action, compute what the Bayesian posterior of the belief will look like. Then, compute how much the entropy of the belief decreases, compared to the entropy before the robot takes the hypothetical action.

   (c) Repeat the above step, but take a hypothetical measurement of 0.

   (d) Compute the expected reduction of entropy related to the hypothetical action, by summing up the entropy reduction when the hypothetical measurement is 1 and when it is 0. It is going to be a weighted sum (what will be the weights in order for this summation to be an expectation?).

7. Take the action that has the largest expected reduction of entropy.

8. Take a new measurement and repeat the process until the entropy of the belief is below a threshold, then the door is at the grid cell with the highest likelihood.

Ergodicity

Expected Information Density

The expected information density in an active sensing problem can be based off of a number of metrics. We could used entropy reduction, information gain maximization, or Fisher information to define the expected utility of our measurement. One way to implement active sensing is to iteratively update the EID and maximize information gain based on the current best estimate, but this is likely to fail if the initial EID is wrong. It is particularly prone to ignoring high information areas when there are multiple peaks in a distribution.

Assuming Gaussian noise, the Fisher Information matrix for estimation of parameter θ given a measurement at a location x from a measurement model $\mathrm{Y}(x, \theta)$ simplifies to

$$\mathcal{I}(x, \theta)_{i,j} = \frac{1}{\sigma^2} \frac{\partial^2 \mathrm{Y}(\theta, x)}{\partial \theta_i \partial \theta_j}.$$

Our estimate of θ is a random variable with prior $p(\theta)$, so we take our expected information density as the expectation over the Fisher information with probability $p(\theta)$ such that,

$$\phi_{i,j}(x) = \frac{1}{\sigma^2} \int_{\theta_i} \int_{\theta_j} \frac{\partial^2 \mathrm{Y}(\theta, x)}{\partial \theta_i \partial \theta_j} p(\theta_i, \theta_j) d\theta_j d\theta_i.$$

We can now use the same optimality metrics (trace, eigenvalue, determinant) that we could on the Fisher information to define the EID. If we choose to use D-optimality, the EID is

$$EID(x) = det\, \phi(x).$$

Definitions of Ergodicity

Ergodicity carries slightly different but related definitions in other fields. If you do an internet search of ergodicity or ergodic theory, you might find:

- **Markov Chains:** A state in a Markov chain is ergodic if you have a nonzero probability of exiting the state and the probability of eventually returning is 1. If all states are ergodic, the chain is ergodic.
- **Signal Analysis:** An process is ergodic if the time average of a signal is equal to the average across an ensemble if signals. In other words, the signal is ergodic if the statistics of the signal matches the statistics of the ensemble. Therefore, each signal is representative of the process, and its possible to estimate the statistics of the ensemble from any one signal.

- **Dynamic Systems:** A system is ergodic if it has the same behavior averaged over time as it does averaged over the space of system states.

When we use the term ergodicity in this class, we are specifically referring to the ergodicity of a trajectory $x(t)$ with respect to a distribution $\phi(x)$. Specifically we say a trajectory is perfectly ergodic with respect to a distribution if the amount of time spent a neighborhood $\mathcal{N}$ of the state space is proportional to the spatial distribution in the neighborhood $\int_{\mathcal{N}} \phi(s)ds$. This is related to the signal analysis definition in that we are trying to create trajectories that are representative of a reference distribution.

Measurement

How might we measure this using information theory? Markov chains test eigenvalues, which clearly isn't an option for trajectories over continuous space. The Kullback-Leibler divergence could probably be used to compare the relative entropy of two signals. Given the experimental mean and variance of two Gaussian signals we could easily compute the relative entropy to measure the degree to which one represents the other. This has a couple of downfalls. First the K-L divergence isn't symmetric. Second, $x(t)$ takes on only one state at each time t, and as a consequence D_{KL} between $x(t)$ and $\phi(x)$ will be generally infinite. Roughly speaking, this is because the variables transverse to the trajectory at each time are being projected onto the trajectory, indicating perfect knowledge and therefore infinite information.

In our earlier lectures, we looked at controlling trajectories based on the L_2 of the error such that we measured the distance between two trajectories as

$$J(x(t)) = \int ||x_d(t) - x(t)||_Q dt.$$

However, this sort of subtraction only works when the two things we want to compare are over the same domain. Instead, we will need a different norm or a way to represent the trajectory and distribution in a different vector space.

Defining a Metric

Let's create a metric by starting with our definition of ergodicity. We said that a trajectory is ergodic when the time spend in a neighborhood (i.e. a subset of the domain) is proportional to the spatial distribution in that neighborhood. Suppose we define a spherical set $B(s, r)$ centered at s with radius r and an indicator function $I_{(s,r)}(y)$

which is equal to one inside the sphere and 0 otherwise. We can then define the average time spent in the set $B(s,r)$ as

$$d^t(s,r) = \frac{1}{t}\int_0^t I_{(s,r)}(x(\tau))d\tau.$$

The measure of the distribution on the same set is given by

$$\bar{\phi}(s,r) = \int_U \phi(y) I_{(s,r)}(y)dy.$$

If the trajectory is ergodic, then $\lim_{t\to\infty} d^t(s,r) = \bar{\phi}(s,r)$ for any pair of (s,r). If this is true for any pair, it must also be true for the infinite sum of these pairs, so one could quantify how far the time averages on these sets are from their spatial averages by taking the integral

$$E^2(t) = \int_0^R \int_U (d^t(s,r) - \phi(s,r))^2 ds dr,\ R > 0.$$

Let's consider instead an alternative way to represent the trajectory, by constructing a distribution,

$$C(x) = \frac{1}{t}\int_0^t \delta(x - x(\tau))d\tau,$$

representing the spatial statistics of the trajectory $x(t)$. Recall that Dirac delta functions fulfill all of the properties of a probability density function, but as was discussed infotaxis, they take on a singular value with infinite information. The inner product $\langle C, f\rangle$ of this distribution and any function is defined as

$$\langle C, f\rangle = \frac{1}{t}\int_0^t f(x(\tau))d\tau.$$

So we could rewrite average time spent in $B(s,r)$ as $d^t(s,r) = \langle C, I_{(s,r)}\rangle$. This allows us to write the Fourier coefficients of the spatial statistics of the trajectory as

$$c_k = \langle C, F_k\rangle = \frac{1}{T}\int_0^T F_k(x(t))dt,$$

using Fourier basis functions of the form,

$$F_k(x(t)) = \frac{1}{h_k}\prod_{i=1}^{n} cos\left(\frac{k_i\pi}{L_i}x_i(t)\right).$$

At each time, the state $x(t)$ is n-dimensional and the subscript k is multi-index over the coefficients of the multi-dimensional Fourier transform. The normalizing factor h_k ensures that F_k is an orthonormal basis and L_i is a measure of the length of the dimension. In 2 dimensions, h_k takes the form,

$$h_k = \left(\int_0^{L_1}\int_0^{L_2} cos^2(\frac{k_1\pi}{L_1}x_1)cos^2(\frac{k_2\pi}{L_2}x_2)dx_1dx_2\right)^{1/2}.$$

Using the same basis functions, we can compute the coefficients of the spatial distribution,

$$\phi_k = \langle \phi(x), F_k \rangle = \int_X \phi(x) F_k(x) dx$$

. Now, the Fourier representations of C and ϕ are in the same vector space called a Sobolev space which also happens to form a Hilbert space. The important thing to take away from this is the a vector space have a particular notion of an inner product or norm to define the distance between two elements in the vector space. The distance between C and $\phi(x)$ as given by the Sobolev space norm of the negative index H^{-s} is

$$\varepsilon(t) = \sum_{k_1=0}^{K} ... \sum_{k_n=0}^{K} \Lambda_k |c_k - \phi_k|^2$$

The coefficient $\Lambda_k = (1 + ||k||^2)^{-s}$ where $s = \frac{n+1}{2}$ places larger weights on lower frequency information.

It turns out that requiring ε to approach zero is equivalent to requiring the time average of the Fourier basis function to converge to the spatial averages of the Fourier basis functions. Additionally, it can be shown that $\varepsilon(t)$ and $E(t)$ are equivalent metrics, since there exists bounded constants such that

$$C_1 \varepsilon(t) \leq E^2(t) \leq C_2 \varepsilon(t).$$

Similar metrics can be derived for ergodicity using other basis functions such as wavelets. However, the primary reason for using this metric with Fourier basis functions is that it is differentiable with respect to $x(t)$. This will be useful when we want to actually define a cost on this metric in the next lecture.

A Geometry Perspective of Ergodic Metric

What is a distance?

We start by asking the question: what is a distance? We often take this concept as granted, but defining distance might be trickier than we think.

A distance is a function. It is defined over a topological space, such as the Euclidean space, and it maps any two points in the space to a non-negative real number. Denote the space as $\mathcal{S}$, we can define as a function to be:

$$d : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}_0^+ \tag{8}$$

Furthermore, a valid distance metric should satisfy the following four properties:

1. (Identity) The distance between two identical points should be zero:

$$d(s, s) = 0, \forall s \in \mathcal{S} \tag{9}$$

2. (Symmetry) The order of the two points should not affect the distance:

$$d(s_1, s_2) = d(s_2, s_1), \forall s_1, s_2 \in \mathcal{S} \tag{10}$$

3. (Positivity) The distance between two distinct points should always be larger than zero:

$$d(s_1, s_2) > 0, \forall s_1 \neq s_2 \in \mathcal{S} \tag{11}$$

4. (Triangle inequality) Given three arbitrary points, the following inequality holds:

$$d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3) \tag{12}$$

   The equality holds if and only if s_3 can be represented as a linear combination of s_1 and s_2.

One of the most commonly used distance metric is Euclidean distance. Given two 3D points $s_1 = [x_1, y_1, z_1]$ and $s_2 = [x_2, y_2, z_2]$, the Euclidean distance be defined as:

$$d(s_1, s_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{13}$$

But why does it look like this? There are two explanations, both of them involve the concept of *inner product*, so let's start with that.

Similar to distance, an inner product is a mapping that maps two arbitrary points of a space to a non-negative real number, it is often denoted as a set of angle brackets:

$$\langle \cdot, \cdot \rangle : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}_0^+ \tag{14}$$

Again, it must satisfy some properties:

1. (Non-negativity) For two arbitrary points, we have:

$$\langle s_1, s_2 \rangle \geq 0, \forall s_1, s_2 \in \mathcal{S} \tag{15}$$

   The equality holds if and only if $s_1 = \mathbf{0}$ or $s_2 = \mathbf{0}$

2. (Symmetry) $\langle s_1, s_2 \rangle = \langle s_2, s_1 \rangle \forall s_1, s_2 \in \mathcal{S}$

3. (Linearity) $\langle s_1, a \cdot s_2 + b \cdot s_3 \rangle = \langle s_1, a \cdot s_2 \rangle + \langle s_1, b \cdot s_3 \rangle = a \cdot \langle s_1, s_2 \rangle + b \cdot \langle s_1, s_3 \rangle$

One important reason that we care about inner product is that it leads to the definition of *norm*, which is a mapping that maps a point to a non-negative real number. For example, in the a 3-dimensional Euclidean space, the inner product of two points (vectors) $s_1 = [x_1, y_1, z_1]$ and $s_2 = [x_2, y_2, z_2]$ is defined as:

$$\langle s_1, s_2 \rangle = x_1 x_2 + y_1 y_2 + z_1 z_2 \tag{16}$$

And the Euclidean norm of a point is defined as the square root of the inner product of the point with itself:

$$|s_1| = \sqrt{x_1^2 + y_1^2 + z_1^2} \tag{17}$$

More importantly, the Euclidean norm is the distance between the point and the origin of the space. This further leads to the definition of Euclidean distance, which is the Euclidean norm of the subtraction between two points:

$$d(s_1, s_2) = |s_1 - s_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{18}$$

However, this is not the full story. The last myth here is, when we represent a 3D point through its xyz coordinate $s_1 = [x_1, y_1, z_1]$, what do we actually mean? A topological space is defined on top of a set of orthonormal bases. For an n-dimensional space, the orthonormal bases are a set of n points (vectors) $\{e_1, \ldots, e_n\} \in \mathcal{S}$ with the following properties:

1. (Normality) $\langle e_i, e_i \rangle = 1, \forall i \in [1, \ldots, n]$

2. (Orthogonality) $\langle e_i, e_j \rangle = 0, \forall i \neq j \in [1, \ldots, n]$

The orthonormal bases define the coordinate system of a space, which allows us to use a set of numbers to represent a point in the space. For the xyz coordinate in an 3D Euclidean space, the most commonly used orthonormal bases are $e_x = [1, 0, 0], e_y = [0, 1, 0], e_z = [0, 0, 1]$, the point can then be represented as the linear combination of the orthonormal bases weighed on the coordinate values:

$$s_1 = [x_1, y_1, z_1] = x_1 \cdot e_x + y_1 \cdot e_y + z_1 \cdot e_z \tag{19}$$

This further implies that, if we define a different set of orthonormal bases, denoted as $\{e'_x, e'_y, e'_z\}$, we can find the coordinates on the new orthonormal bases by taking the inner product between the original coordinate and the new orthonormal bases:

$$\begin{aligned} s_1 = [x_1, y_1, z_1] &= x_1 \cdot e_x + y_1 \cdot e_y + z_1 \cdot e_z && (20) \\ &= \langle s_1, e'_x \rangle \cdot e'_x + \langle s_1, e'_y \rangle \cdot e'_y + \langle s_1, e'_z \rangle \cdot e'_z && (21) \end{aligned}$$

This is also often called as coordinate transformation.

Based on the concept of orthonormal bases, we can extend the definition of Euclidean distance to an arbitrary set of orthonormal bases:

$$d_{e_x, e_y, e_z}(s_1, s_2) = \sqrt{(\langle s_1, e_x \rangle - \langle s_2, e_x \rangle)^2 + (\langle s_1, e_y \rangle - \langle s_2, e_y \rangle)^2 + (\langle s_1, e_z \rangle - \langle s_2, e_z \rangle)^2} \tag{22}$$

Now, after we re-derive the Euclidean distance metric over an arbitrary set of orthonormal bases, we are looking beyond the finite-dimensional Euclidean space, instead we are looking to define the Euclidean distance between two functions in a similar way.

Distance between two functions

We first define the space of all functions that map from the same space to the same other space $\mathcal{F} = \{f | f : \mathcal{X} \mapsto \mathcal{Y}\}$. In order to define the distance between any two functions f_1, f_2 from the space, we start with extending the fundamental concepts such as inner product to this function space.

The inner product between two functions is defined as:

$$\langle f_1, f_2 \rangle = \int_{\mathcal{X}} f_1(x) f_2(x) dx \tag{23}$$

With this definition, for a set of orthonormal bases $\{g_1, g_2, g_3, \dots\}$ of the function space, it must satisfy the following properties:

1. $\int_{\mathcal{X}} g_i(x) g_i(x) dx = 1, \forall i$
2. $\int_{\mathcal{X}} g_i(x) g_j(x) dx = 0, \forall i \neq j$

Note that since the function space is infinite-dimensional, the orthonormal bases will have an infinite number of basis functions as well.

Now, we can follow the derivation of the distance metric in metric, the one based on the inner product with orthonormal bases, to derive the distance between two functions:

$$d(f_1, f_2) = \sqrt{\sum_{i=1}^{\infty} (\langle f_1, g_i \rangle - \langle f_2, g_i \rangle)^2} \tag{24}$$

The more familiar format of the distance metric between two functions is as follow:

$$d(f_1, f_2) = \sqrt{\int_{\mathcal{X}} (f_1(x) - f_2(x))^2 dx} \tag{25}$$

This format is derived on the top of a set of point-wise Dirac delta basis functions defined as:

$$g_i(x) = \delta_{s_i}(x) = \begin{cases} +\infty, & \text{if } x{=}s_i, s_i \in \mathcal{X} \\ 0, & \text{otherwise} \end{cases} \tag{26}$$

Distance between a trajectory and a distribution

The real question we are interested in is to compute the distance between a trajectory and a probability distribution. We denote the trajectory as a mapping $s : [0, T] \mapsto \mathcal{X}$ and the probability density function $p : \mathcal{X} \mapsto \mathbb{R}_0^+$.

The first step is to transform the trajectory, which is now a function of time, to the same domain as the probability density function, which is a function of space. To do so, we define the empirical distribution of the trajectory using the Dirac delta function:

$$\Phi_s(x) = \frac{1}{T} \int_0^T \delta_{s(t)}(x) dt \tag{27}$$

The Dirac delta function has a very nice property regarding inner product, given an arbitrary delta function $\delta_s(x)$ and an arbitrary function $f(x)$, we have:

$$\langle \delta_s(x), f(x) \rangle = f(s) \tag{28}$$

Based on this property, the inner product between the empirical distribution of the trajectory $\Phi_s(x)$ and an arbitrary function $f(x)$ can

be computed as closed-form:

$$\langle \Phi_s(x), f(x) \rangle = \int_{\mathcal{X}} \Phi_s(x) f(x) dx \tag{29}$$

$$= \int_{\mathcal{X}} \left(\frac{1}{T} \int_0^T \delta_{s(t)}(x) dt \right) f(x) dx \tag{30}$$

$$= \frac{1}{T} \int_0^T \left(\int_{\mathcal{X}} \delta_{s(t)}(x) f(x) dx \right) dt \tag{31}$$

$$= \frac{1}{T} \int_0^T f(s(t)) dt \tag{32}$$

This means, given a trajectory and a orthonormal basis function, we can compute the coefficient of the orthonormal function in closed-form. This gives us the distance between a trajectory and a distribution, given a set of orthonormal basis functions $\{g_1(x), g_2(x), g_3(x), \dots\}$:

$$d(s(t), p(x)) = \sqrt{\sum_{i=1}^{\infty} \Big(\langle \Phi_s(x), g_i(x) \rangle - \langle p(x), g_i(x) \rangle \Big)^2} \tag{33}$$

Normalized Fourier basis functions

So far our discussion assumes we have a set of orthonormal basis functions, now we define a set of orthonormal functions that enable efficient numerical approximation, based on Fourier basis functions.

Define a function $f : \mathcal{X} \mapsto \mathbb{R}$, where $\mathcal{X} = [L_1^l, L_1^u] \times \cdots \times [L_d^l, L_d^u] \subset \mathbb{R}^d$ is a $d-$dimensional rectangular space, L_i^l and L_i^u are the lower and upper bound for the d-th dimension, respectively. For any function defined as above, we can define the following normalized Fourier basis functions as the bases for orthonormal decomposition:

$$f_{\mathbf{k}}(x) = \frac{1}{h_{\mathbf{k}}} \prod_{i=1}^{d} \cos \left(\bar{k}_i (x_i - L_i^l) \right) \tag{34}$$

where

$$x = (x_1, x_2, \dots, x_N) \in \mathbb{R}^d, \quad \mathbf{k} = [k_1, \cdots, k_d] \in [0, 1, 2, \cdots, \mathbf{K}]^d \subset \mathbb{N}^d$$

$$\bar{k}_i = \frac{k_i \pi}{L_i^u - L_i^l}, \quad h_{\mathbf{k}} = \left(\prod_{i=1}^{d} \frac{L_i^u - L_i^l}{2} \right)^{\frac{1}{2}}$$

Control Synthesis for Ergodic Objectives

Defining a cost

In this lecture we will start to bring together the concepts we learned about continuous time optimal control and what we have discussed with regards to information measures and their corresponding objective functions. To begin, we reintroduce a system with dynamics,

$$\dot{x} = f(x(t), u(t)),\ x(0) = x_0.$$

Previously, we have had cost functions of the form,

$$J = \int_0^t l(x,u)dt = \int_0^t (x_d - x)^T Q(x_d - x) + u^T R u\, dt,$$

where we minimize J with respect to u subject to the constraints of the dynamics by optimizing a descent directions at each step of an iterative process. When we want our system to maximize information instead of minimizing error the form our cost function takes changes. If we use the ergodic metric described in the previous lecture our cost is

$$\begin{aligned} J(x(t), u(t)) &= q\,\varepsilon(x(t)) + \int_0^T u(t)^T R u(t) dt \\ &= q \sum_{k_1=0}^{K} \dots \sum_{k_n=0}^{K} \Lambda_k \left(\frac{1}{T} \int_0^T F_k(x(t)) dt - \phi_k \right)^2 + \int_0^T u(t)^T R u(t) dt. \end{aligned}$$

For brevity, we will using a single summation and multi-index k to represent the nested summations of the squared term over $K+1$ basis functions for each of the n dimensions in the rest of the notation in this lecture.

Derivative of an Ergodic Objective

Now, let's start to fill in an iterative algorithm that finds the minimizer of that cost function. First, we need a terminal condition for our algorithm. A natural choice for this is to use the first derivative of the cost—the necessary condition for a minimizer. We want the partial derivative of J with respect to $\xi = (x(t), u(t))$ to be less than some small value ϵ. More precisely, our terminal condition should be $||DJ(\xi) \cdot \zeta|| > \epsilon$—the directional derivative of the cost function. We calculate $DJ(\xi) \cdot \zeta$ by taking the directional derivative in the direction $\zeta = (z(t), v(t))$.

$$\begin{aligned} \frac{d}{d\epsilon} J(\xi + \epsilon\zeta)|_{\epsilon=0} = \frac{d}{d\epsilon} \Bigg[q \sum_{k=0}^{K} \Lambda_k \left(\frac{1}{T} \int_0^T F_k(x(s) + \epsilon z(s)) dt - \phi_k \right)^2 + \\ \int_0^T (u(t) + \epsilon v(t))^T R (u(t) + \epsilon v(t)) dt \Bigg]_{\epsilon=0} \end{aligned}$$

$$\frac{d}{d\epsilon}J(\xi+\epsilon\zeta)|_{\epsilon=0} = \left[q\sum_{k=0}^{K}\Lambda_k\left[2\left(\frac{1}{T}\int_0^T F_k(x(s)+\epsilon z(s))ds-\phi_k\right)\cdot\int_0^T\frac{1}{T}DF_k(x(t)+\epsilon z(t))z(t)dt\right]\right.$$
$$\left.+\int_0^T(u(t)+\epsilon v(t))^T R\cdot v(t)dt\right]_{\epsilon=0}$$

$$DJ(\xi)\cdot\zeta = q\sum_{k=0}^{K}\Lambda_k\left[2\left(\frac{1}{T}\int_0^T F_k(x(s))ds-\phi_k\right)\cdot\int_0^T\frac{1}{T}DF_k(x(t))z(t)dt\right]+\int_0^T u(t)^T R\cdot v(t)dt$$

This Euclidean norm (or weighted norm) of this expression is the stopping criteria for our optimization. When this derivative is close to zero, we have found a local extrema.

Finding the Direction of Steepest Descent

Now that we have a termination criteria, we can establish what we should be doing during each iteration. If we want to do something like gradient descent, we need to find a direction ζ that maximizes the change in J subject to some cost on the magnitude of ζ. Back in lecture 5, we wrote the general form of the derivative of the cost on state error as

$$DJ(\xi)\cdot\zeta = \int_0^T Dl(\xi)\cdot\zeta dt = \int_0^T D_x l(\xi)z(t)+D_u l(\xi)v(t)dt.$$

We can rewrite our expression for the derivative so that it is in this single integral form by pulling the experession in parentheses into the second integral and switching the order of our summation and integral to get

$$DJ(\xi)\cdot\zeta = \int_0^T q\sum_{k=0}^{K}\Lambda_k\left[2\left(\frac{1}{T}\int_0^T F_k(x(s))ds-\phi_k\right)\cdot\frac{1}{T}DF_k(x(t))\right]\cdot z(t)+u(t)^T R\cdot v(t)\ dt.$$

Defining $a^T(t) = q\sum_{k=0}^{K}\Lambda_k\left[2\left(\frac{1}{T}\int_0^T F_k(x(s))ds-\phi_k\right)\cdot\frac{1}{T}DF_k(x(t))\right]$ and $b^T(t) = u(t)^T R$, we can write the derivative in the form,

$$DJ(\xi)\cdot\zeta = \int_0^T a^T(t)\cdot z(t)+b^T(t)\cdot v(t)dt.$$

We can now choose ζ as a minimizer of a quadratic cost

$$g(\zeta) = \int_0^T Dl(\xi)\cdot\zeta\,dt+\frac{1}{2}\langle\zeta(t),\zeta(t)\rangle,$$

subject to constraint $\dot{z(t)} = A(t)z(t)+B(t)v(t)$, where $A(t) = D_1 f(x,u)$ and $B(t) = D_2 f(x,u)$. We have not yet established what the quadratic operator placing a cost on the magnitude of ζ should be. First, we know that since ζ is a function of time, we should integrate it. Second, ζ is part of a vector space, so we get to define a norm.

This could be the Euclidean 2-norm or the weighted 2-norm where we define a Q and R such that they are symmetric positive semi-definite matrices. The Hamiltonian for this system is then

$$H = a^T z + b^T v + \frac{1}{2}(z^T Q z + v^T R v) + p^T(Az + Bv)$$

and applying Pontryagin's maximum principle we can solve Hamilton's equations

$$\begin{aligned} \dot{z} &= Az + Bv \\ \dot{p} &= -a - Qz - A^T p \\ 0 &= b + Rv + B^T p \end{aligned}$$

to get the optimal direction ζ. As in our iLQR solution, if we assume $p = Pz + r$, we can solve the standard Riccati differential equation to find our direction of steepest descent.

$$\begin{aligned} \dot{z}(t) &= A(t)z(t) + B(t)v(t),\ z(0) = 0, \\ v(t) &= R(t)^{-1}B(t)^T P(t)z(t) - R(t)^{-1}B(t)^T r(t) - R(t)^{-1}b(t) \\ \dot{P}(t) &= P(t)B(t)R(t)^{-1}B(t)^T P(t) - Q(t) - P(t)A(t) + A(t)^T P(t) \\ \dot{r}(t) &= -(A(t) - B(t)R(t)^{-1}B(t)^T P(t))^T r(t) - a(t) + P(t)B(t)R(t)^{-1}b(t) \end{aligned}$$

Applying the Update to ξ

To determine the step size to take in the optimal direction, one can use an Armijo line search, or use second order methods to determine the appropriate step size. Alternatively, one can just choose a constant small value γ as the step size for every iterate. This is used to update the control and trajectory according to

$$\begin{aligned} u_{i+1} &= u_i + \gamma v \\ x_{i+1} &= x(0) + \int_0^T f(x_{i+1}(t), u_{i+1}(t))dt \end{aligned}$$

When Should ϕ be updated?

In our pseudocode, we would never update ϕ because unlike infotaxis, we are computing an open-loop continuous trajectory. However, if we applied our ergodic cost to feedback controller or a receding horizon controller such as Sequential Action Controller, we have a choice of what we can do after each new measurement is taken. There are a couple of downsides to updating the coefficients ϕ_k of the distribution. First, it is computationally expensive to recompute the coefficients. There is also a risk that updating too frequently under high uncertainty or modeling error will lead to overreactive exploration strategies that perform poorly. In short, the answer might be to almost never recompute the distribution coefficients.

Algorithm 4: Ergodic Exploration

Initialize the distribution coefficients ϕ_k, ξ_0, $DJ(\xi_i) \cdot \zeta_i$.

$i = 0$
while $||DJ(\xi_i) \cdot \zeta_i|| > \epsilon$ **do**
  i=i+1
  Calculate descent direction:
    $\zeta_i = \arg\min_{\zeta} DJ(\xi_i) \cdot \zeta_i + \frac{1}{2}\langle \zeta_i, \zeta_i \rangle$
  Choose step size γ_i using Armijo, second order methods, or a small value.
  Update the control, $u_i = u_{i-1} + \gamma_i v_i$.
  Update the trajectory, $x_i = x(0) + \int_0^T (f(x_i, u_i))dt$
end while

Nonparametric vs. Parametric Estimation/Models

What are the differences?. Trick question! They both have some parameters that define the estimation/models (we will be using this language interchangeably). Basically, what makes parametric models "parametric" is that assume we have a finite set of parameters θ that is able to describe future predictions x given some data set (or observed data) $\mathcal{D}$. Nonparametric models, in contrast, do not make this assumption on the finite set of parameters θ. In fact, nonparametric models assume that the data distribution $\mathcal{D}$ can not be modeled by a finite set of parameters, but rather an *infinite* set of parameters θ. You might be thinking "how can anyone even compute this infinite set of parameters?". In this lecture (and the notes) we will overview Gaussian processes as a way to define this infinite set of parameters as a function. Moreover, we will discuss a Bayesian way of thinking about nonparametric models for predictions based on observed data.

Before we begin, we will list some of the differences between parametric and nonparametric estimation from a set of data.

Parametric

- Models data set with a *finite* set of parameters θ
- Bounded model complexity
- Rigid structure (linear parametric models can only model at best observed data with linear structure)
- Unbounded data (does not scale with the amount of data)

Nonparametric

- Infinite set of parameters model data

- Complexity grows with the number of data points
- Is very difficult to calculate when the observed data set is very large
- Very flexible

Preliminary Information

Before diving into the wonderful world of Gaussian processes, we must first lay down some knowledge that will be useful in understanding Gaussian processes and other nonparametric Bayesian analysis. To start, a Gaussian process is a Bayesian method of generating models from some observed set of data $\mathcal{D}$. This type of method generates a posterior predictive distribution, or rather, given the set of data $\mathcal{D}$, what is the most likely prediction $\tilde{y}$ given a test value $\tilde{x}$ knowing the existing data x? This is a type of regression problem where we will be focusing on the use of Gaussian processes for making future predictions from existing data. Another way you can think about this is that Gaussian processes (and Bayesian methods) provide you with a way of dealing with new data given that you already have some old data and want to make use of it in some way.

The Gaussian (Normal) Distribution and some of its properties

Let us define $x \in \mathbb{R}^n$ to be a random variable with mean $\mu \in \mathbb{R}^n$ and covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$.[9] This random variable is normally distributed (or is Gaussian distributed) if the probability function is of the form

$$p(x \mid \mu, \sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right). \quad (35)$$

[9] Σ is symmetric positive definite.

In many textbooks, one writes samples from this distribution as $x \sim \mathcal{N}(\mu, \Sigma)$.

Below are some properties of Gaussian distributions:

Normalization The integral over the whole space x is equal to 1.

$$\int_x p(x \mid \mu, \Sigma) dx = 1 \quad (36)$$

Marginalization Let's say we have the probability $p(x_1, x_2 \mid \mu, \Sigma)$ from the set where $x = [x_1, x_2]$. We can create the two distributions

$$p(x_1) = \int_{x_2} p(x_1, x_2 \mid \mu, \Sigma) dx_2 \quad (37)$$

and

$$p(x_2) = \int_{x_1} p(x_1, x_2 \mid \mu, \Sigma) dx_1 \quad (38)$$

by integrating out the variables x_1 or x_2. Note that $x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$ and $x_2 \sim \mathcal{N}(\mu_2, \Sigma_{22})$ where

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \quad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \tag{39}$$

Conditioning We can define conditional distributions as

$$p(x_1 \mid x_2) = \frac{p(x_1, x_2 \mid \mu, \Sigma)}{\int_{x_1} p(x_1, x_2 \mid \mu, \Sigma) dx_1} \tag{40}$$

$$p(x_2 \mid x_1) = \frac{p(x_1, x_2 \mid \mu, \Sigma)}{\int_{x_2} p(x_1, x_2 \mid \mu, \Sigma) dx_2} \tag{41}$$

which are also Gaussian distributions with

$$x_1 \mid x_2 \sim \mathcal{N}(\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}) \tag{42}$$

$$x_2 \mid x_1 \sim \mathcal{N}(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}). \tag{43}$$

Summation Two independent Gaussian random variables are also Gaussian.

The following section, we will be using these identities for regression and generating predictions based on observed data.

Linear Regression with the Gaussian Distribution and Maximum Likelihood Estimation

Now that we have set up the preliminary equations we will now concern ourselves with predictions of pairwise data (i.e., input x gives us measurement y with noise, can we predict the measurement $\tilde{y}$ from a new point $\tilde{x}$?).

Let us define an observed data set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ of the pair x_i, y_i (not to be confused with the previous notation) where x_i is input to the measurement y_i with some noise. Here, $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$ (we can always write this observation is higher dimensions but for clarity we will leave this is 1-D). For now, let's assume that we are doing linear regression and the pairwise data is assumed to be generated from the equation

$$y_i = \theta^\top x_i + \epsilon \tag{44}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is the noise from our sensors. Here, the only knowledge we have is the observed data set $\mathcal{D}$ and an assumption on the noise. We do not know the noise value ϵ, but we do know the

variance of the noise σ^2. Using the fact that we assume the noise is normally distributed, we can write a Gaussian likelihood function

$$p(y_i \mid x_i, \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \theta^\top x_i)^2}{2\sigma^2}\right) \tag{45}$$

over the parameters θ. Here Equation (45) provides us with a likelihood distribution over the data set $\mathcal{D}$ which we can use to figure out what is the most likely set of parameters θ that generated our data set. This is essentially a maximum likelihood estimation where we *extremize* the likelihood function to solve for the parameters θ.

Before we start to solve what the parameters θ might be, let us rewrite Equation (45) as the full likelihood over the whole data set of size N:

$$p(\mathcal{D} \mid \theta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \theta^\top x_i)^2}{2\sigma^2}\right). \tag{46}$$

Great! Now that we have this, we want to ... that is right! Take a derivative of Equation (46) with respect to θ. Note that the derivative is going to be a bit painful as there is a product over all the data points and calculus has chain rule which makes the derivative very tedious. We can do a trick that will make this optimization a bit easier. To start, let us take the log of Equation (46):

$$\begin{aligned} \log p(\mathcal{D} \mid \theta) &= \sum_{i=1}^{N} \log p(y_i \mid x_i, \theta) \\ &= -\frac{N}{2} \log 2\pi - N \log \sigma - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \theta^\top x_i)^2. \end{aligned} \tag{47}$$

Equation (47) is known as the log-likelihood function. Basically it is exactly what the name suggests, the log of the likelihood function which for us is a Gaussian which means it takes the form of the very familiar linear regression. Taking the derivative of the log-likelihood function and setting it to zero gives

$$\frac{\partial}{\partial \theta} \log p(\mathcal{D} \mid \theta) = -\frac{1}{\sigma^2} \sum_{i=1}^{N} (y_i - \theta^\top x_i) x_i = 0. \tag{48}$$

We can rewrite this derivative in vector notation to make this easier

$$-\frac{1}{\sigma^2} (\vec{y} - X\theta)^\top X = -\frac{1}{\sigma^2} \vec{y}^\top X + \frac{1}{\sigma^2} \theta^\top X^\top X = 0 \tag{49}$$

where

$$X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_N^\top \end{bmatrix} \in \mathbb{R}^{N \times n}, \text{ and } \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N. \tag{50}$$

Note that you can't just divide out X since it is a matrix and can be zero. The solution to θ in Equation (49) is

$$\theta^\top = \vec{y}^\top X(X^\top X)^{-1} \tag{51}$$

which for those interested $(X^\top X)^{-1}$ is the pseudoinverse of X. We can take the transpose and move things around, but you should get the idea of maximum likelihood estimation and working directly with likelihood functions from this example.

Quick Example

Quick example before we move on to Bayesian optimization. Let us consider the function

$$y = \begin{bmatrix} 10.0 \\ 1.0 \end{bmatrix}^T x \tag{52}$$

where $x \in \mathbb{R}^2$ and $y \in \mathbb{R}^1$. Let's say our observed data set is

$$\mathcal{D} = \{([1,2]^\top, 11.98), ([4,10]^\top, 49.93), ([-1,1]^\top, -8.97)\} \tag{53}$$

with assumed Gaussian noise with $\sigma = 0.1$. We can put things in the right format with

$$X = \begin{bmatrix} 1 & 2 \\ 4 & 10 \\ -1 & 1 \end{bmatrix} \text{ and } \vec{y} = \begin{bmatrix} 11.98 \\ 49.93 \\ -8.97 \end{bmatrix}. \tag{54}$$

Plugging this into Equation (51) gives

$$\theta = [9.97349282, 1.00358852]^\top. \tag{55}$$

Not exact, but close to the actual underlying function. With significantly more data, this approximation becomes closer and closer to the true equation that generated the data. Note that this same analysis can be done for $y \in \mathbb{R}^m$, but I will leave that up to you as an exercise.

Bayesian Optimization for Linear Regression

We are going to switch gears and look at this same problem of linear regression from a Bayesian view point. In the Bayesian world, we often assume that we have some sort of prior over the parameters θ (this can be uniform or normally distributed). For convenience, let's define $\theta \sim \mathcal{N}(0, \sigma_\theta^2 \mathbf{I})$. We want to do the same that we did with Maximum Likelihood estimation, but using the fact that we have a prior on the parameters θ and that we have at our disposal Bayes' rule to use old data to make predictions about new data.

We can define a posterior on the parameter using Bayes' rule as

$$p(\theta \mid \mathcal{D}) = \frac{p(\theta)p(\mathcal{D} \mid \theta)}{\int_\theta p(\theta)p(\mathcal{D} \mid \theta)d\theta} = \frac{p(\theta)\prod p(y_i \mid x_i, \theta)}{\int_\theta p(\theta)\prod p(y_i \mid x_i, \theta)d\theta}. \tag{56}$$

Using this parameter posterior, based on the observed data $\mathcal{D}$, we can test new points $\tilde{x}$ for outputs $\tilde{y}$ using a posterior predictive distribution defined as

$$p(\tilde{y} \mid \tilde{x}, \mathcal{D}) = \int_\theta p(\tilde{y} \mid \tilde{x}, \theta)p(\theta \mid \mathcal{D})d\theta. \tag{57}$$

This is kind of hard and annoying to calculate. This can also be quite difficult if the space of parameters θ is quite large. As it turns out, by making the Gaussian assumption, we can explicitly write

$$\theta \mid \mathcal{D} \sim \mathcal{N}\left(\frac{1}{\sigma^2}A^{-1}X^\top \vec{y}, A^{-1}\right) \tag{58}$$

$$\tilde{y} \mid \tilde{x}, \mathcal{D} \sim \mathcal{N}\left(\frac{1}{\sigma^2}\tilde{x}^\top A^{-1}X^\top \vec{y}, \tilde{x}^\top A^{-1}\tilde{x} + \sigma^2\right) \tag{59}$$

where $A = \frac{1}{\sigma^2}X^\top X + \frac{1}{\sigma_\theta^2}\mathbf{I}$ and

$$X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_N^\top \end{bmatrix} \in \mathbb{R}^{N\times n}, \text{ and } \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}. \in \mathbb{R}^N. \tag{60}$$

For those interested in the proof, I would recommend referring to: *Carl E. Rasmussen and Christopher K. I. Williams. Gaussian Processes for Machine Learning. MIT Press, 2006. Online: http://www.gaussianprocess.org/gpml/* which is a great source for learning more about Gaussian Processes and more about the nitty gritty details.

So what is different in this solution than with the Maximum likelihood method? Correct! There is an extra parameter that shows up in the mean value of θ. Moreover, θ is now a distribution which we can sample over. Interestingly, this solution is identical to the Ridge Regression formulation of linear regression where there is an additional regularization term (here that is $\frac{1}{\sigma_\theta^2}\mathbf{I}$).

Quick example

Let's revisit the example from before with

$$y = \begin{bmatrix} 10.0 \\ 1.0 \end{bmatrix}^T x \tag{61}$$

and our observed data set is

$$\mathcal{D} = \{([1,2]^\top, 11.98), ([4,10]^\top, 49.93), ([-1,1]^\top, -8.97)\} \tag{62}$$

with assumed Gaussian noise with $\sigma = 0.1$. We can put things in the same format as before:

$$X = \begin{bmatrix} 1 & 2 \\ 4 & 10 \\ -1 & 1 \end{bmatrix} \text{ and } \vec{y} = \begin{bmatrix} 11.98 \\ 49.93 \\ -8.97 \end{bmatrix}. \tag{63}$$

Let's say we are pretty unsure of the parameter θ , so we choose $\sigma_\theta = 10.0$. Plugging this information into Equation (58) gives us

$$\theta = [9.97315669, 1.00370931]^\top \tag{64}$$

which is very close to what we originally calculated with Maximum Likelihood estimation. So why do some prefer this method? Well, we can calculate how unsure we are of the estimated parameter θ from the data set. Specifically the term A^{-1} is our variance term which in this case is

$$A^{-1} = \begin{bmatrix} 0.00502383 & -0.00196169 \\ -0.00196169 & 0.00086123 \end{bmatrix}. \tag{65}$$

This is a pretty powerful tool. We can use the data to provide a confidence intervals over the parameter space.

In the case of active learning, we can use this to guide a controller to minimize the variance as much as possible by looking at where in the input vector x is there the most variance.

Moving onto Function Approximation

So now we are going to get into the interesting case of evaluating nonlinear function mappings from $x \rightarrow y$ using this Bayesian approach. We will first consider just looking at a finite set of functions and fitting the functions to old data points to estimate new data points. We will then show that this analysis has some very interesting structure that can be used to estimate all possible functions (a.k.a. the Gaussian Process).

Let us consider the same set of data $\mathcal{D}$. Instead of fitting the parameters θ, we want to fit over functions $f(x)$

$$y_i = f(x_i) + \epsilon \tag{66}$$

where $f(x) = \phi(x)^\top \theta$ such that $\phi(x)$ is a function that maps $\mathbb{R}^n \rightarrow \mathbb{R}^m$. In other words, $\phi(x)$ maps input data points to an output of m dimensional features. The motivation for this is that we want to use Bayesian logic to choose the best function over the features that best explains the data. Using the analysis from before for Bayesian linear regression, we can write

$$\tilde{f} \mid \tilde{x}, X, \vec{y} \sim \mathcal{N}(\frac{1}{\sigma^2}\phi(\tilde{x})^\top A^{-1}\Phi\vec{y}, \phi(\tilde{x})^\top A^{-1}\phi(\tilde{x})) \tag{67}$$

where

$$\Phi = \Big[\phi(x_1), \phi(x_2), \ldots, \phi(x_N)\Big] \in \mathbb{R}^{m \times N} \tag{68}$$

and $A = \frac{1}{\sigma^2}\Phi\Phi^\top + \frac{1}{\sigma_\theta^2}\mathbf{I}$. Note that A is a matrix of size $m \times m$ and we need to invert it. If the number of features m is very large, this inversion is probably not the easiest thing to achieve. However, we can use a bit of manipulation to get the following equivalent expression

$$\tilde{f} \mid \tilde{x}, X, \vec{y} \sim \mathcal{N}(\tilde{\phi}^\top\Sigma_\theta\Phi(K+\sigma^2\mathbf{I})^{-1}\vec{y}, \tilde{\phi}^\top\Sigma_\theta\tilde{\phi} - \tilde{\phi}^\top\Sigma_\theta\Phi(K+\sigma^2\mathbf{I})^{-1}\Phi^\top\Sigma_\theta\tilde{\phi}) \tag{69}$$

where $\Sigma_\theta = \frac{1}{\sigma_\theta^2}\mathbf{I}$, $\tilde{\phi} = \phi(\tilde{x})$, and $K = \Phi^\top\Sigma_\theta\Phi$. Now, the matrix $K \in \mathbb{R}^{N \times N}$ which means that our inversion is now determined by the number of data points! Thus, you can have an arbitrarily large number of basis functions without needing to have a large inversion.

If you take a look at the Bayesian linear regression, you can see that in fact the equations are very similar where the only difference is that you are transforming the input data x with some function $\phi(x)$. Note that often this function is considered a set of features that possibly describe the nonlinearities in the data.

Quick example

Let's take the example of the data set $\mathcal{D}$:

x data		y data
-0.1	-0.9	-0.80291152
0.1	-0.1	-0.09101267
-0.2	-0.3	-0.33012962
-0.6	0.2	0.07270773
-0.4	-0.5	-0.55961684
0.2	0.1	0.15158343

generated from the equation

$$y = \begin{bmatrix} x_0 & \sin(x_1) \end{bmatrix} \begin{bmatrix} 0.2 \\ 1.0 \end{bmatrix} \tag{70}$$

where the noise $\epsilon \sim \mathcal{N}(0, 0.01)$ is normally distributed and the parameters $\theta \sim \mathcal{N}(0, 1.0)$ are also normally distributed.

What is really great about Bayesian optimization is that we do not need to know the functions that generate the data, but we can try out any set of functions and get a good estimate. For us, let's use

$$\phi_{\text{test}}(x) = \begin{bmatrix} x_0 & x_1 & \sin(x_0) & \sin(x_1) \end{bmatrix}^\top. \tag{71}$$

Note that we are overloading the notation slightly (the subscript in the function does not mean the data points x_i, y_i, but rather the individual elements of $x_i \in \mathbb{R}^2$. The Φ matrix is then

$$\Phi = \begin{bmatrix} x_{0,0} & x_{1,0} & x_{2,0} & \dots & x_{N,0} \\ x_{0,1} & x_{1,1} & x_{2,1} & \dots & x_{N,1} \\ \sin(x_{0,0}) & \sin(x_{1,0}) & \sin(x_{2,0}) & \dots & \sin(x_{N,0}) \\ \sin(x_{0,1}) & \sin(x_{1,1}) & \sin(x_{2,1}) & \dots & \sin(x_{N,1}) \end{bmatrix} \tag{72}$$

where $x_{i,j}$ is the j^{th} element of the i^{th} data point. If we look at Equation (67), we can see that $\theta = A^{-1}\Phi\vec{y}/\sigma^2$ which when we calculate it becomes

$$\theta = \begin{bmatrix} 0.20571431 & 0.03706383 & 0.0014112 & 0.95667461 \end{bmatrix}^\top. \tag{73}$$

Notice that the terms that do show up in the original equation have weights that are similar where the other terms have small non-zero values. Even though those equations do not show up, this optimization method still uses those additional weights to make predictions. If we have a larger data set, these values would be closer to zero. Modifying σ_θ would also change our result for θ quite a bit. However, the resulting predictions are very close to the original data set if one were to test them out (you should try this our and convince yourself). However, predictions elsewhere not within the data set might be better or worse.

Now that we are able to use these tools to estimate parametric models, you should be able to create distributions and beliefs over

data sets and functions. Thus, it is possible to use these distributions for ergodic control for instance, where you want to be ergodic with respect to the likelihood of a measurement. Similarly, one can compute the entropy of the distributions and use this for active sensing/learning.

Larger Function Spaces (even Infinite)

Now we are moving on to the case where $\phi(x)$ is very large (and possibly unknown). Essentially the Gaussian Process.

What do you notice about the feature vector $\phi(x)$ in Equation (69)? It is a bit hard to see but in Equation (69), the feature vectors always enter as an inner product in the various forms $\Phi^\top \Sigma_\theta \Phi$, $\phi(\tilde{x})^\top \Sigma_\theta \Phi$, or $\phi(\tilde{x})^\top \Sigma_\theta \phi(\tilde{x})$. Let us define this inner product as $k(x, x') = \phi(x)^\top \Sigma_\theta \phi(x')$ as a kernel function (also known as a covariance function).

If one knew what the kernel function $k(x, x')$ was to begin with, one could simply just compute that value instead of calculating the feature vectors and make the prediction calculations with less computations. This is known as the kernel trick and there exists quite a few kernels that approximate an *infinite* set of feature vectors. Just what are the conditions for a kernel you ask? Well, there is a theorem called Mercer's theorem that provides the conditions for which an inner product over a set of feature vectors constitutes as a kernel function. The most common one is the Radial Basis Function (RBF) or squared exponential kernel, and believe it or not, this approximates an infinite series of features given by

$$k(x, x') = \exp\left(-\frac{1}{2}|x - x'|^2\right). \tag{74}$$

The Gaussian Process Regressor Let us define the Gaussian process where the random variable is the function $f(x)$ where

$$m(x) = \mathbb{E}\left[f(x)\right] \tag{75}$$

is the mean function and

$$k(x, x') = \mathbb{E}\left[(f(x) - m(x))(f(x') - m(x'))\right] \tag{76}$$

is the covariance function. The Gaussian process is then written as

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')). \tag{77}$$

If we look at the previous example for $f(x) = \phi(x)^\top \theta$ where $\theta \sim \mathcal{N}(0, \Sigma_\theta)$ we have for the mean and covariance

$$\mathbb{E}\left[f(x)\right] = \phi(x)^\top \mathbb{E}\left[\theta\right] = 0, \tag{78}$$

$$\mathbb{E}\left[f(x)f(x')\right] = \phi(x)^\top \mathbb{E}\left[\theta\theta^\top\right]\phi(x') = \phi(x)^\top \Sigma_\theta \phi(x') = k(x, x'). \quad (79)$$

We can see that a reasonable assumption is to have a zero mean and the kernel function defined as our covariance. We can use this to generate predictions by specifying the joint distribution from the data set $\mathcal{D}$ as

$$\begin{bmatrix} f \\ \tilde{f} \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma^2\mathbf{I} & \vec{k}(\tilde{x}) \\ \vec{k}(\tilde{x})^\top & k(\tilde{x},\tilde{x}) \end{bmatrix}\right) \quad (80)$$

where $K(X, X)$ is a matrix of $N \times N$ covariance functions $k(x, x')$ evaluated at the data points or more formally

$$K(X,X) = \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \dots & k(x_1,x_N) \\ k(x_2,x_1) & k(x_2,x_2) & \dots & k(x_1,x_N) \\ \vdots & \ddots & & \\ k(x_N,x_1) & & & k(x_N,x_N) \end{bmatrix}, \quad (81)$$

$\vec{k}(\tilde{x}) \in \mathbb{R}^N$ is the vector of covariance functions $k(x, x')$ where each x is evaluated at the data point and x' is the new test point $\tilde{x}$:

$$\vec{k}(\tilde{x}) = \begin{bmatrix} k(x_1,\tilde{x}) & k(x_2,\tilde{x}) & \dots k(x_N,\tilde{x}) \end{bmatrix}^\top \quad (82)$$

As done before, we can condition on the previous data points to acquire a predictive posterior distribution where

$$\tilde{f} \mid X, \vec{y}, \tilde{x} \sim \mathcal{N}(\vec{k}(\tilde{x})^\top (K(X,X) + \sigma^2\mathbf{I})^{-1}\vec{y}, k(\tilde{x},\tilde{x}) - \vec{k}(\tilde{x})^\top (K + \sigma^2\mathbf{I})^{-1}\vec{k}(\tilde{x}).). \quad (83)$$

Thus, predictions are of the form

$$\tilde{f} = \vec{k}(\tilde{x})^\top (K + \sigma^2\mathbf{I})^{-1}\vec{y} \quad (84)$$

and the variance

$$\mathbb{V}[\tilde{f}] = k(\tilde{x},\tilde{x}) - \vec{k}(\tilde{x})^\top (K + \sigma^2\mathbf{I})^{-1}\vec{k}(\tilde{x}). \quad (85)$$

Notice that this has the exact form as Equation 69) which makes sense since we have only redefined the inner product over a larger function space as the kernel function. As an exercise, you should be able to compute the Gaussian process for a set of data points using the Radial Basis function (and another kernel function) and provide the variance given the data points. One of the great things about Gaussian processes is that you can sample directly on the mean prediction and generate various different function samples that approximate the data. Moreover, you can use the variance as a measure for how certain a robot can be about a particular region of space or a parameter (the choice of what the data represents is with you).

Appendix

Sequential Action Control

(notes originally drafted by K. Fitzsimons in 2018)

Motivation

Up until this point we have discussed how one might generate a curve of control values $u(t)$ over an entire time horizon, which works well in the simulated systems of the homework. However, if you are interested in applying optimal control on a real system, we have two issues to contend with. The first is that there is virtually no way that our model of the system and the environment will capture the uncertainties and perturbations that the robot will encounter. One might hope that these perturbations are small enough that our open loop solution for $u(t)$ will work, or we can recalculate $u(t)$ based on current sensor(state) feedback. This brings us to our second problem, the *time* it takes to calculate these control curves. On our real system using our iterative methods to find an optimizer over the entire time horizon may not be fast enough for us to recover from a perturbation. In the worst case, our new $u(t)$ will be completely outdated by the time we have found it. To work around this, we can choose to only calculate the open loop control for a very short time horizon into the future or we can insist that our hardware have sufficient computational speed.

Alternatively, we can reframe the problem so that we are only looking for what our next control action should be in the next t_s seconds, where $t_s = \frac{1}{SamplingRate}$. Rather than looking for a curve over some window into the future we can search for a single action to apply in that window. In other words, we will search for a control action defined by a control vector, u_2^*, an application time τ, and an application duration, λ. What we will find is that there is an analytic solution for a schedule of optimal actions $u_2^*(\tau)$, allowing us to perform a simple line search over the scalar τ and choose λ (also scalar) by enforcing a descent condition.

Hybrid System Dynamics

Suppose we have a system of the form:

$$\dot{x} = f(x(t), u(t)).$$

The system may be nonlinear with respect to the state, but we will restrict ourselves to cases where, f is linear with respect to the control,

satisfying control-affine form.

$$f(t, x(t), u(t)) = g(x(t)) + h(x(t))u(t)$$

Given a nominal control, $u = u_1$, the dynamics of the system can be described by $f_1 \triangleq f(x(t), u_1(t))$. Suppose that we want to take a single *short* action—perturbing the nominal dynamics—that improves some cost of the form:

$$J_1 = \int_{t_0}^{t_f} l_1(x(t))dt + m(x(t_f))$$

This will make the control look like,

$$u(t) = \begin{cases} u_1(t) & t < \tau,\ t > \tau + \lambda \\ u_2^*(\tau) & \tau \leq t \leq \tau + \lambda \end{cases}$$

We now have to find the control vector $(u_2^*(\tau))$, the time τ when we want to apply the control, and the (short) control duration λ. The system can now be described by,

$$x(t) = \begin{cases} x(0) + \int_{t_0}^{t} f_1(x(s))ds & t_0 \leq t < \tau \\ x(0) + \int_{t_0}^{\tau} f_1(x(s))ds + \int_{\tau}^{t} f_2(x(s))ds & \tau \leq t \leq \tau + \lambda \\ x(0) + \int_{t_0}^{\tau} f_1(x(s))ds + \int_{\tau}^{\tau+\lambda} f_2(x(s))ds + \int_{\tau+\lambda}^{t} f_1(x(s))ds & t > \tau + \lambda \end{cases},$$

where $f_2 \triangleq f(x(t), u_2^*(\tau))$.

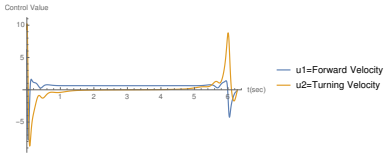


Figure 31: A Control signal for the differential drive vehicle generate by iLQR.

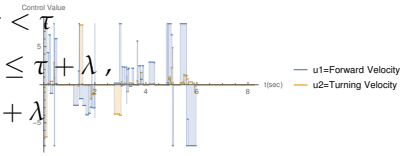


Figure 32: A Control signal for the differential drive vehicle generate by Sequential Action Control(right) and iLQR(left).

The Mode Insertion Gradient

Rather than searching for the optimal control vector, application time, and control duration simultaneously, we begin by assuming that τ and f_2 are known. Now we can model the change in cost of a particular (τ, f_2) by taking the derivative of J_1 with respect to $\lambda \to 0^+$.

$$\frac{dJ_1}{d\lambda^+} = \int_{t_0}^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} \frac{\partial x(t)}{\partial \lambda^+} dt$$

What does $\frac{\partial x(t)}{\partial \lambda^+}$ look like? Calculating this from the integral equation for $x(t)$ we get:

$$\underbrace{\frac{\partial x(t)}{\partial \lambda^+}}_{z} = \begin{cases} 0 & t_0 \leq t < \tau \\ 0 & \tau \leq t \leq \tau + \lambda \\ \lim_{\lambda \to 0^+} \underbrace{f_2(x(\tau+\lambda)) - f_1(x(\tau+\lambda))}_{z_\tau} + & t > \tau + \lambda \\ \quad \int_{\tau}^{\tau+\lambda} Df_2(x(s)) \underbrace{\frac{\partial x(t)}{\partial \lambda^+}}_{z} ds + \int_{\tau+\lambda}^{t} Df_1(x(s)) \underbrace{\frac{\partial x(t)}{\partial \lambda^+}}_{z} ds & \end{cases} \quad (86)$$

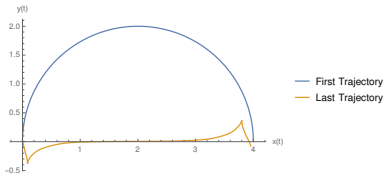


Figure 33: A trajectory for the differential drive vehicle generate by iLQR.

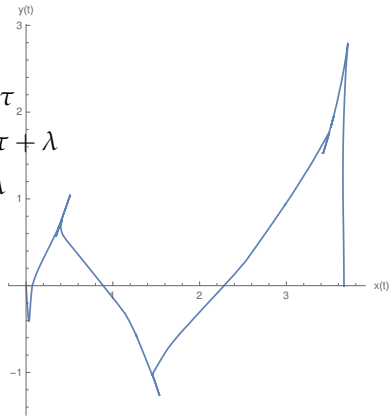


Figure 34: A trajectory for the differential drive vehicle generate by Sequential Action Control.

Applying the limit $\lambda \to 0^+$, this equation for $\frac{\partial x(t)}{\partial \lambda^+}$ is the integral form of differential equation for an LTV system:

$$\dot{z} = \begin{cases} 0 & t_0 \leq t < \tau \\ Az(t) & t \geq \tau \end{cases} \quad z(\tau) = f_2(x(\tau)) - f_1(x(\tau)),\ A = Df_1(x(t)) \tag{87}$$

Now we can write,

$$\frac{dJ_1}{d\lambda^+} = \int_{t_0}^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} \frac{\partial x(t)}{\partial \lambda^+} dt = \int_{\tau}^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} z(t) dt,$$

where z is the solution to the linear differential equation above. Now remember that we haven't actually chosen a specific τ or $u_2^*(\tau)$, so for each choice of these two we would need to solve the differential equation and integrate. Instead let's use the state transition matrix for $z(t)$ in the LTV system in the $\frac{dJ_1}{d\lambda^+}$.

$$\int_{\tau}^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} z(t) dt = \int_{\tau}^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} \Phi(t,\tau)[f_2(x(\tau)) - f_1(x(\tau))] \tag{88}$$

$$= \underbrace{\int_{\tau}^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} \Phi(t,\tau) dt}_{p^T} [f_2(x(\tau)) - f_1(x(\tau))] \tag{89}$$

The second step is possible because $[f_2(x(\tau)) - f_1(x(\tau))]$ does not depend on time. The other term is the convolution integral of a linear affine system running backwards in time. If we call the state variable for this convolution equation $p = \int_{\tau}^{t_f} \Phi^T(t,\tau) \left[\frac{\partial l_1(x(t))}{\partial x(t)}\right]^T dt$, we get that

$$\dot{p} = -A^T p - \frac{\partial l_1(x(t))}{\partial x(t)}^T$$

The final condition for this differential equation is $p(t_f) = 0_{N\times 1}$ if you have neglected the terminal cost $m(x(t_f))$ in your cost J_1 as we did at the start of this derivation. If we had not we would have $p(t_f) = \nabla m(x(t_f))$. With this, we get the *mode insertion gradient*

$$\frac{dJ_1}{d\lambda^+}(\tau, u_2^*(\tau)) = p^T[f_2(x(\tau)) - f_1(x(\tau))].$$

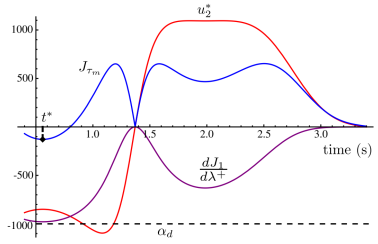


Figure 35: A schedule of optimal actions drives the mode insertion gradient towards α_d. The optimal application time τ is when $\frac{dJ_1}{d\lambda^+}$ is most negative. (From: Ansari & Murphey 2016)

Schedule of Optimal Actions

You can think of the mode insertion gradient as a measure of the first-order sensitivity of the cost J_1 to a perturbation of the control $u_2^*(\tau)$ at time τ for an infinitesimal duration. We want to choose a control value and time that maximizes the cost sensitivity in the negative direction. We can do this by first finding a schedule of action values $u_2(\tau)$ that minimizes,

$$l_2(\tau, u_2(\tau)) = \frac{1}{2}[\frac{dJ_1}{d\lambda^+}(t, u_2(\tau)) - \alpha_d]^2 + \frac{1}{2}||u_2(\tau)||_R^2,$$

where α_d is the desired sensitivity.

If we evaluate the optimal schedule of actions u_2^* at any time $\tau \in (t_0, t_f)$ that value should minimize l_2, so the entire schedule must also minimize the infinite sum J_2 of the costs associated with each possible value of $u_2^*(\tau)$.

$$J_2 = \int_{t_0}^{t_f} l_2(t, u_2(t))dt$$

In $\frac{dJ_1}{d\lambda^+}$, x depends only on u_1, so unlike the minimization of ζ in Lecture 3, there is no constraint associated with the minimization of J_2. We can now take the directional derivative and set it equal to zero to find our minimizer. This gives us an *analytic solution* for u_2^*.

$$\begin{aligned} DJ_2 \cdot \eta(t) &= \frac{d}{d\epsilon} \int_{t_0}^{t_f} l_2(t, u_2(t) + \epsilon\eta(t))dt|_{\epsilon=0} && (90) \\ &= \int_{t_0}^{t_f} \frac{\partial l_2(t, u_2^*(t))}{\partial u_2(t)} \eta(t)dt = 0 \ \forall \eta && (91) \\ \frac{\partial l_2(t, u_2^*(t))}{\partial u_2(t)} &= 0 && (92) \\ \frac{\partial l_2}{\partial u_2} &= (p^T h(x)[u_2^* - u_1] - \alpha_d)p^T h(x) + u_2^{*T} R = 0 && (93) \\ u_2^*(\tau) &= (\Lambda + R^T)^{-1}[\Lambda u_1 + h(x)^T p \ \alpha_d] && (94) \end{aligned}$$

where $\Lambda \triangleq h(x)^T p p^T h(x)$. This gives use a schedule of actions from which to choose the next control. Although we included a cost on the magnitude of u_2 this doesn't guarantee that the control vector will obey the saturation limits of the system. However, it turns out that control vectors computed from the analytic solution are affine with respect to α_d and linear when $u_1 = 0$, so scaling α_d produces actions that are scaled linearly. This implies that if any component of the control vector violated the saturation limits, we can choose a new α_d to satisfy the constraint. This may produce overly conservative controls when only one element of a multidimensional control vector violates a constraint, but we can also apply scaling element-wise while still producing an action capable of reducing the cost J_1.

Application Time and Duration

We still need to choose our application time τ and a duration λ. We can use a very small sampling time t_s and choose to just apply the next control based on the schedule at $\tau = t_0 + t_s$ for t_s seconds. Alternatively, we can apply the entire schedule as a descent direction. Finally, we can find τ using an optimization, which we will discuss below.

We now introduce one final cost function to find the time τ at which we can be most effective in applying our single control. Again,

we want to minimize $\frac{dJ_1}{d\lambda^+}$ with an additional cost on the control effort and the cost of waiting to apply a control.

$$J_\tau = ||u_2^*(\tau)|| + \frac{dJ_1}{d\lambda^+} + (t - t_0)^\beta,\ \beta > 0$$

Depending on the coding language you are using, there are many ways to find the optimizer. Something like MATLAB's fmincon or Mathematica's Minimize would work here. You can also be creative and create your own minimization function.

Up until this point, we have assumed we are applying control for an infinitely short duration, but assuming we want this to work in the real world we will have to choose a $\lambda > 0$. We do this using a line search with a simple descent condition. Alternatively, you can just choose a very small t_s and apply control

Algorithm Design Choices

For finite durations, our model of the change in cost is locally described by,

$$\Delta J_1 \approx \frac{dJ_1}{d\lambda^+}(\tau, u_2^*(\tau))\lambda.$$

Our optimization over u_2^* regulates the mode insertion gradient such that,

$$\frac{dJ_1}{d\lambda^+}(\tau, u_2^*(\tau)) \approx \alpha_d$$

Therefore, these two parameters allow us to specify the degree of change provided by each action.

$$\Delta J_1 \approx \alpha_d \lambda$$

A high α_d will lead to aggressive actions by the controller that are often saturated and make dramatic changes in the cost. While any negative number should work, it is often helpful to specify α_d as a feedback law,

$$\alpha_d = \gamma J_{1,init},$$

where $\gamma \in [-15, -1]$ works well. Given a specified α_d and a corresponding $u_2^*(\tau)$ we can choose λ using a line search with a descent condition requiring a minimum ΔJ_{min}, which can be specified as a (negative) percentage of the initial value of J_1.

Finally, β in the optimization of τ is often in the range of $(0.5, 2)$ but any $\beta > 0$ will work.

SAC Pseudocode

Algorithm 5: Sequential Action Control

Initialize α_d, minimum change in cost ΔJ_{min}, current time t_{curr}, default control duration Δt_{init}, nominal control u_1, scale factor $\omega \in (0,1)$, predictive horizon T, sampling time t_s, the max time for iterative control calculations t_{calc}, and the max backtracking iterations k_{max}.

while $t_{curr} < \infty$ **do**
  $(t_0, t_f) = (t_{curr}, t_{curr} + T)$
  Use feedback to initialize $x_{init} = x(t_0)$
  Simulate (x, p) from f_1 for $t \in [t_0, t_f]$
  Compute initial cost $J_{1,init}$
  Specify α_d
  Compute u_2^* from (x, p) using Theorem 1:
    $u_2^* = (\Lambda + R^T)^{-1}[\Lambda u_1 + h(x)^T p \alpha_d]$
    $\Lambda \triangleq h(x)^T p p^T h(x)$
  Specify/search for time, $\tau > t_0 + t_{calc}$, to apply u_2^*
  Saturate $u_2^*(\tau)$
  Initialize $k = 0,\ J_{1,new} = \infty$
  while $J_{1,new} - J_{1,init} > \Delta J_{min}$ **and** $k \leq k_{max}$ **do**
    $\lambda = \omega^k \Delta t_{init}$
    $(\tau_0, \tau_f) = (\tau - \frac{\lambda}{2}, \tau + \frac{\lambda}{2})$
    Re-simulate x applying f_2 for $t \in [\tau_0, \tau_f]$
    Compute new cost $J_{1,new}$
    $k = k + 1$
  end while
  $u_1(t) = u_2^*(\tau)\ \forall\ t \in [\tau_0, \tau_f] \cap [t_0 + t_{calc}, t_0 + t_s + t_{calc}]$
  Send updated u_1 to robot
  while $t_{curr} < t_0 + t_s$ **do**
    Wait()
  end while
end while