16-745 Lecture Notes

tshankar

March 22, 2021

LECTURE 1

1 Logistics

If you haven't checked out these course logistics, here are some useful links.

- The course syllabus can be found on Canvas - `https://canvas.cmu.edu/files/5927463/download?download_frd=1`.
- If you haven't filled out the course survey, please do so here - `https://forms.gle/8MW6STDPHENtWH9Z6`.
- Log into the course slack for all course communication - the workspace is `16-745optimalcontrol.slack.com`.
- Also check out the course github. We'll be using this for distributing and collecting homeworks from you, and posting lecture notes, etc. `https://github.com/Optimal-Control-16-745`.

2 Fundamentals

We first describe some fundamental concepts that we'll use throughout the course.

2.1 Continuous Time Dynamics

The most general way of describing smooth dynamical systems is via a dynamics function:

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

Here, $x \in \mathbb{R}^n$ describes the state of the system, $u \in \mathbb{R}^n$ is the control input provided to the system, and f, the dynamics function, specifies how the system evolves with the application of control inputs. $\dot{x}$ provides the derivatives (with respect to time) of state.

For a mechanical system, the state x is usually described as:

$$x = \begin{bmatrix} q \\ v \end{bmatrix} \tag{2}$$

where q describes the configuration of the system, and v describes the velocity.

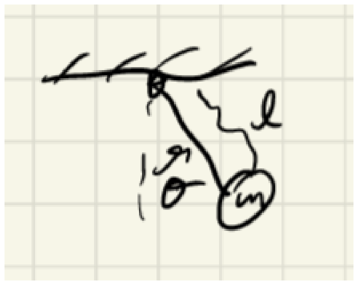

Figure 1: Simple pendulum system

2.1.1 Validity of Assumptions

Note that the configuration isn't necessarily a vector, and the velocities are not necessarily derivatives of configuration! Also, remember that this description of the system is valid only when the dynamics are smooth. This is broken when, for instance, the system experiences contacts.

2.1.2 Example

Consider the example of a simple pendulum fig. 1. The system dynamics can be captured in the following equation:

$$ml^2\ddot{\theta} + mgl\sin(\theta) = \tau \tag{3}$$

Here, the configuration q is given by the pendulum angle θ, the velocity v is specified by $\dot{theta}$, and the control inputs u are given by the torque applied to the system τ.
Thus the state x is:

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \tag{4}$$

The velocity $\dot{x}$ is:

$$\dot{x} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l}\sin\theta + \frac{1}{ml^2}u \end{bmatrix} \tag{5}$$

Here, the system dynamics $f(x, u)$ are given as

$$f(x, u) = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l}\sin\theta + \frac{1}{ml^2}u \end{bmatrix} \tag{6}$$

The manifold of state here is described as $x \in \mathbb{S}^1 \times \mathbb{R}$, a cylinder.

2.2 Control-Affine Systems

Many systems (mechanical systems in particular) can be defined as a *control-affine system*. This is a specific form of the general dynamics described above, where the control inputs affect the system via an affine matrix G.

$$\dot{x} = f_0(x) + G(x)u \tag{7}$$

For instance, the pendulum system can be described as a control-affine system, where

$$f_0(x) = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l}\sin(\theta) \end{bmatrix} \quad G(x) = \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} \tag{8}$$

In driftless systems, $f_0(x) = 0$. One can also convert systems to control-affine systems, by adding original control inputs u to state, and setting the modified control inputs (of the control-affine system) to be the derviatives of u.

2.3 Manipulator Dynamics

Another common form of expressing dynamics is -

$$M(q)\dot{v} + C(q, u) = B(q)u \tag{9}$$

Where $M(q)$ is the mass matrix of the system, $C(q, u)$ is the dynamics bias (including Coriolis terms and gravity), and $B(q)$ is the control input jacobian. As usual, q is the configuration and v is the velocity. Here, the change in configuration $\dot{q}$:

$$\dot{q} = G(q)v \tag{10}$$

describe the kinematics of the system.

$$\dot{x} = f(x, u) = \begin{bmatrix} G(q)v \\ -M(q)^{-1}(B(q)u - C) \end{bmatrix} \tag{11}$$

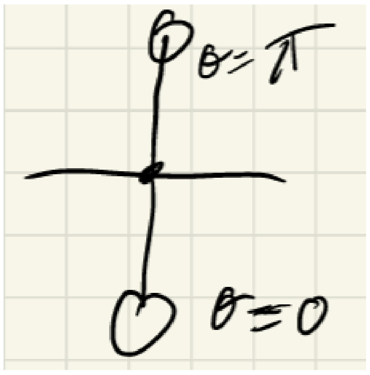

Figure 2: Equilibria points of simple pendulum system

In the pendulum system, the mass matrix $M(q) = ml^2$, $C(q,u) = ql\sin(\theta)$, $B = I$, $G = I$.

All mechanical systems can be described in this form; this is because this form is a different way of expressing the Euler-Lagrange equation for:

$$L = 1/2\ u^T M(q)u - V(q) \tag{12}$$

2.4 Linear Systems

Linear systems are common way of expressing control problems and designing controllers, since we know how to solve linear systems, and they are relatively simple to deal with.

$$\dot{x} = A(t)x + B(t)u \tag{13}$$

These linear systems are either linearly time invariant, when matrices $A(t)$ and $B(t)$ are constant over time. They are linearly time varying systems otherwise.

We typically approximate non-linear systems with linear systems by linearizing the system around the current state. Then -

$$\dot{x} = f(x,u) \tag{14}$$

where $A = \frac{\partial f}{\partial x}$ and $B = \frac{\partial f}{\partial u}$.

2.5 Equilibria

We also want to understand equilibria - i.e. the point where a system will remain at rest. Correspondingly -

$$\dot{x} = f(x,u) = 0 \tag{15}$$

Algebraically, this is described by roots of the dynamics equation.

In the pendulum example (without a control input) fig. 2 -

$$\dot{x} = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l}\sin(\theta) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{16}$$

The roots of this occur when $\theta = 0$ and $\dot{\theta} = 0, \pi$.

2.5.1 First control problem

Lets consider a small control problem. Let's try to move the equilibrium point of the pendulum, by applying an appropriate control input. In this case -

$$\dot{x} = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l}\sin(\frac{\pi}{2}) + \frac{1}{ml^2}u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{17}$$

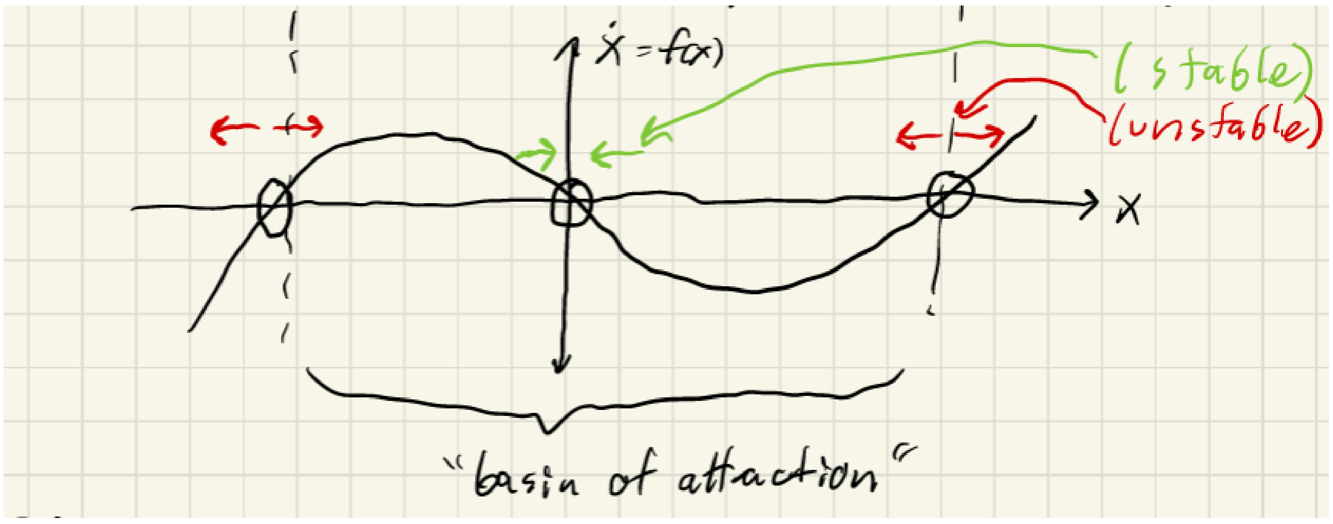

Figure 3: Stability of equilibria of a 1-D system.

Solving for u, we have:

$$\frac{1}{ml^2}u = -\frac{g}{l}\sin(\frac{\pi}{2}) \implies u = mgl \tag{18}$$

2.6 Stability of Equilibria

We'd also like to understand when a system will return to its equilibrium point upon being perturbed. Consider a 1-D system, $x \in \mathbb{R}$, as in fig. 3.

In this case, when $\frac{\partial f}{\partial x} < 0$, the system is stable, because it gets pushed back to the equilibrium point by virtue of the dynamics. Conversely, for $\frac{\partial f}{\partial x} > 0$, the system is unstable, because it gets pushed away from the equilibrium point. The region of x such that $\frac{\partial f}{\partial x} < 0$ is called the basin of attraction of a system.

In higher dimensions, $\frac{\partial f}{\partial x}$ is a Jacobian matrix. To study stability of such a system, we can take an Eigen decomposition of the Jacobian (decoupling it into n 1-D systems). If the real-component of each of the eigven values of the Jacobian is less than 0, the system is stable. Mathematically, the system is stable if

$$\text{Re}\big[eig(\frac{\partial f}{\partial x})\big] < 0 \tag{19}$$

and unstable otherwise.

In our pendulum example,

$$f(x) = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l}\sin(\theta) \end{bmatrix} \implies \frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l}\cos(\theta) & 0 \end{bmatrix} \tag{20}$$

Consider the equilibrium point $\theta = \pi$.

$$\implies \frac{\partial f}{\partial x}\Big|_{\theta=\pi} = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{bmatrix} \implies eig(\frac{\partial f}{\partial x}\Big|_{\theta=\pi}) = \pm\sqrt{\frac{g}{l}} \tag{21}$$

Since one eigenvalue is negative, the equibilrium point $\theta = \pi$ is unstable. In contrast, at the equilibrium point $\theta = 0$.

$$\implies \frac{\partial f}{\partial x}\Big|_{\theta=0} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{bmatrix} \implies eig(\frac{\partial f}{\partial x}\Big|_{\theta=\pi}) = 0 \pm i\sqrt{\frac{g}{l}} \tag{22}$$

In this case, the real part of the eigenvalue is 0. In this purely imaginary eigenvalue case, the system is called marginally stable, and experiences undamped oscillations. Adding damping to the systems (such as applying control inputs $u = -Kd\dot{\theta}$, results in strictly negative eigenvalues, leading to a stable system.

3 References

Remember to check out [2] and [3] for this course and these topics!

References

[1] J. Nocedal et al. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer, 2006. ISBN: 9780387303031. URL: `https://books.google.com/books?id=eNlPAAAAMAAJ`.

[2] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control.* Wiley, 2005. ISBN: 9780471649908. URL: `https://books.google.com/books?id=A0OXDwAAQBAJ`.

[3] S.H. Strogatz and M. Dichter. *Nonlinear Dynamics and Chaos, 2nd ed. SET with Student Solutions Manual.* Studies in Nonlinearity. Avalon Publishing, 2016. ISBN: 9780813350844. URL: `https://books.google.com/books?id=vUWhDAEACAAJ`.

LECTURE 2

4 Agenda

This lecture covers the following topics!

- Continuous ODEs, and how to use discrete time simulation for this.
- More on stability.

5 Motivation

In general cases (i.e. when we don't have convenient forms of f), we can't solve $\dot{x} = f(x)$ for $x(t)$. This implies we can't design appropriate controllers either. In these cases, we solve such systems computationally, and need to represent $x(t)$ in discrete-time. Happily, discrete time models can also capture some effects that continuous ODEs cannot capture, and so in some sense are more general than their continuous counterparts. For example, contacts, and other discontinuous events, can be more easily captured by discrete-time models.

6 Discrete-Time Dynamics

Let's consider the explicit form of such discrete-time systems, where the state at the next timestep is determined as:

$$x_{k+1} = f_{\text{discrete}}(x_k, u_k) \tag{23}$$

The simplest discretization that we could use is Forward Euler integration -

$$x_{k+1} = x_k + h f_{\text{continuous}}(x_k, u_k) \tag{24}$$

Where h denotes some step size.
Let's go back to our favorite pendulum example here! If we use $l = m = 1$, $h = 0.1$ or $h = 0.01$. This blows up! (Check out the notebook in the Lecture). Why does it blow up? To answer this, we need to discuss stability of discrete-time systems.

7 Stability of Discrete-Time Systems

Remember, in continuous time, we have that a system is stable when:

$$\text{Re}\big[eig(\frac{\partial f}{\partial x})\big] < 0 \tag{25}$$

Now in discrete time, these dynamics are an in iterated map:

$$x_N = f_{\text{d}}(f_{\text{d}}(f_{\text{d}}...f_{\text{d}}(x_0))) \tag{26}$$

If we linearize this and apply the chain rule:

$$\frac{\partial x_N}{\partial x_0} = \frac{\partial f_{\text{d}}}{\partial x}\frac{\partial f_{\text{d}}}{\partial x}...\frac{\partial f_{\text{d}}}{\partial x}\Big|_{x_0} \tag{27}$$

If A_{d} represents $\frac{\partial f_{\text{d}}}{\partial x}$ around the point of linearization, this A_{d} stays the same over iterations, thus the above expression is:

$$\frac{\partial x_N}{\partial x_0} = \frac{\partial f_{\text{d}}}{\partial x}\frac{\partial f_{\text{d}}}{\partial x}...\frac{\partial f_{\text{d}}}{\partial x}\Big|_{x_0} = A_{\text{d}}^N \tag{28}$$

Let's say the equilibrium point is the origin $x = 0$ (which we can do without loss of generality by a change of coordinates). Thus the stability of this system implies that:

$$\lim_{k \longrightarrow \infty} A_{\text{d}}^k x_0 = 0 \;\; \forall x_0 \tag{29}$$

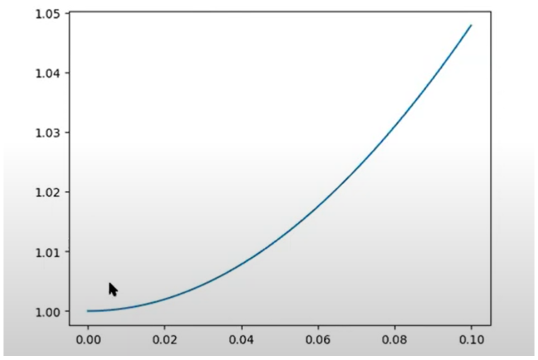

Figure 4: Plot of eigenvalues versus step size.

If this is true $\forall x_0$, this implies A_{d}^k itself must follow:

$$\lim_{k \longrightarrow \infty} A_{\mathrm{d}}^k = 0 \tag{30}$$

This implies that the Eigenvalues of A_{d} must satisfy:

$$|\mathrm{eig}(A_{\mathrm{d}})| < 1 \tag{31}$$

Equivalently, the eigenvalues of A_{d} must lie within the unit circle on the complex plane.
Let's try this out on our example of the pendulum with Forward Euler integration. We have:

$$x_{k+1} = x_k + hf(x_k) = f_{\mathrm{d}}(x_k) \tag{32}$$

We know:

$$A_{\mathrm{d}} = \frac{\partial f_{\mathrm{d}}}{\partial x_k} = I + hA_{\mathrm{continuous}} \tag{33}$$

Remember from last lecture:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l}\cos(\theta) & 0 \end{bmatrix} \tag{34}$$

So:

$$A_{\mathrm{d}} = \frac{\partial f_{\mathrm{d}}}{\partial x_k} = I + h \begin{bmatrix} 0 & 1 \\ -\frac{g}{l}\cos(\theta) & 0 \end{bmatrix} \tag{35}$$

What are the Eigenvalues of this A_{d}?

$$\mathrm{eig}(A_{\mathrm{d}}\big|_{\theta=0}) = 1 \pm 0.313i \tag{36}$$

Since this doesn't lie in the unit circle, this is unstable! If we create a plot of $\mathrm{eig}(A_{\mathrm{d}})$ versus different values of h, we observe the plot: We see the system is marginally stable in the limit of $h \to 0$. This means that for any value of timestep h, the system is going to blow up!

7.1 Take-Away Messages!

The takeaway message from this is -

- Be careful when discretizing ODEs!
- Do a sanity check based on the equilibrium energy behavior (whether there is conservation and / or dissipation of energy).
- This is an artifact of forward Euler integration - it always overshoots the function it is trying to approximate (in this case, it overestimates the acceleration of the system), so avoid it! Especially for undamped systems!

7.2 A better explicit integrator

A better explicit integrator to use is the 4th order Runge-Kutta method, the "industry standard". The intuition of this is that Euler Integration fits a line-segment over each timestep, whereas the 4th order Runge-Kutta method (RK4) fits a cubic polynomial, which is much more expressive, and allows for much better accuracy in capturing the underlying function.
Here's some pseudo code for the RK4 method -

Algorithm 1 4th Order Runge-Kutta Integration

```
1: $x_{k+1} = f_{RK4}(x_k)$
2: $K_1 = f(x_k)$                                   ▷ Evaluate at beginning.
3: $K_2 = f(x_k + \frac{1}{2}hK_1)$                 ▷ Evaluate at midpoint.
4: $K_3 = f(x_k + \frac{1}{2}hK_2)$                 ▷ Evaluate at midpoint.
5: $K_4 = f(x_k + hK_3)$                            ▷ Evaluate at end.
6: $x_{k+1} = x_k + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$   ▷ Evaluate as weighted average of intermediate computations.
```

Let's try this on our Pendulum example. Instead of the explosion we observed before, the eigenvalues are right within the unit circle, and reflect the true continuous system.

7.3 Take-Away Message!

The 4th order Runge-Kutta method provides us with much improved accuracy, that easily outweighs the additional computational cost of 4 function evaluations instead of 1 that forward Euler integration did. That said, even sophisticated integrators have issues, depending on the value of timesteps, etc. So always perform some kind of sanity check!

8 Implicit Form

Another thing to consider is the implicit form of an ODE. For example, the implicit form of a discrete-time dynamical system is:

$$f_{\mathrm{d}}(x_{k+1}, x_k, u_k) = 0 \tag{37}$$

The simplest example of this is:

$$x_{k+1} = x_k + hf(x_{k+1}) \tag{38}$$

which is *Backward Euler Integration*, where we are evaluating the function f at a future time.
How do we simulate this kind of system? You can write this as:

$$f_{\mathrm{d}}(x_{k+1}, x_k, u_k) = x_k + hf(x_{k+1}) - x_{k+1} = 0 \tag{39}$$

You can then treat this as a root-finding problem in x_{k+1}. (We'll spend time on this next week).
In the pendulum example, we obesrve the following -

- We see the opposite energy behavior from forward Euler integration.
- Discretization adds artificial damping to the system, instead of adding energy to the system.
- Backward Euler undershoots the function being approximated.
- While this doesn't reflect the physical process (is unphysical), this effect allows simulators to take big steps, and can be convenient sometimes!
- It is very common to use this in quick and easy low-fi simulators in graphics and robotics.

8.1 Take-Away Message!

- Implicit methods are often "more stable" than their explicit counterparts.
- For forward simulation, solving the implicit equation can be more expensive.
- In many "direct" trajectory optimization methods, they are *not* any more expsensive to use!

9 Discretizing Controls

So far, we've focused on discretizing the state of the system $x(t)$. We also have the control input to the system, $u(t)$ that we need to discretize!
The simplest option here is to set:

$$u(t) = u_k \ t_k \leq t \leq t_{k+1} \tag{40}$$

This is a "zero-order hold", where the control input at time t is simply equal to the control input within the corresponding time interval. This is a piecewise constant control. This is easy to implement, but may require lots of knot points (sample points) to accurately capture a continuous control input $u(t)$.
There are possibly better options, such as:

$$u(t) = u_k + \frac{u_{k+1} - u_k}{h}(t - t_n) \tag{41}$$

This is a "first order hold", or a piecewise *linear* control. The control input is a linear function of the control inputs at the end points of the time interval. This first-order hold can better approximate continuous $u(t)$ with fewer knot points, thus leading to faster solve times, etc. It's not too much more work than the zero-order hold computationally speaking. If you know your system is continuous, use the first order hold! It's also very common to do this, such as classic DIRCOL (direct collocation).
There are other options too -

- We can progressively move to higher-order holds, using higher-order polynomials.
- In many control applications, $u(t)$ is *not* smooth, such as in bang-bang control methods. In these cases, higher-order polynomials are not the best choice, and are not good approximations.

In general, zero-order and first-order holds are the most common ways to discretize control in practice.

LECTURE 3

10 Agenda

This lecture covers some notation that will be useful in representation of matrices. We'll also go over root finding, and minimization techniques, with and without equality constraints. Also remember to check out the lecture recording to get an overview of Homework 1!

11 Notation

Consider a function $f(x) : \mathbb{R}^n \longrightarrow \mathbb{R}$. For this function, we have its derivatives

$$\frac{\partial f}{\partial x} \in \mathbb{R}^n \tag{42}$$

i.e. the derivative $\frac{\partial f}{\partial x}$ is a row vector. This is because the $\frac{\partial f}{\partial x}$ is a linear operator mapping Δx into Δf:

$$f(x + \Delta x) \approx f(x) + \frac{\partial f}{\partial x}\ \Delta x \tag{43}$$

Similarly, given a function $g(y) : \mathbb{R}^m \longrightarrow \mathbb{R}^n$, we have:

$$\frac{\partial g}{\partial y} \in \mathbb{R}^{n \times m} \tag{44}$$

because $g(y + \Delta y) \approx g(y) + \frac{\partial g}{\partial y}\ \Delta y$.
This is important because it makes the chain rule work:

$$f(g(y + \Delta y)) \approx f(g(y)) + \frac{\partial f}{\partial x}\Big|_{g(y)} \frac{\partial g}{\partial y}\Big|_{y}\ \Delta y \tag{45}$$

For convenience, we will also define the gradient:

$$\nabla f(x) = (\frac{\partial f}{\partial x})^T \in \mathbb{R}^{n \times 1} \tag{46}$$

which is a column vector. Further, the second derivative (or the Hessian) is -

$$\nabla^2 f(x) = (\frac{\partial^2 f}{\partial x^2})^T = \frac{\partial}{\partial x}(\nabla f(x)) \in \mathbb{R}^{n \times n} \tag{47}$$

12 Root Finding

Now that we have our notation fixed, let's consider how we can find a "root". Consider a function $f(x)$. We want to find a "root" x^* such that $f(x^*) = 0$. This is closely related to finding the fixed point of a system:

$$f(x^*) = x* \tag{48}$$

Root finding is finding the equilibrium point of a continuous ODE. Fixed points are equilibrium points of *discrete* time versions of these ODEs.

12.1 Newton's Method

: One way to go about root finding is to use Newton's method. Here, we use a Taylor approximation of our function around our initial guess, and solve for x around this guess.

$$f(x + \Delta x) \approx f(x) + \frac{\partial f}{\partial x}\Big|_{x} \Delta x = 0 \implies \Delta x = -(\frac{\partial f}{\partial x})^{-1} f(x) \tag{49}$$

We can then find the root of the system by repeatedly updating our estimate of the root with: $x \leftarrow x + \Delta x$, until the root converges.

12.2 Backward Euler

We can do this using Backward Euler Newton's method; Newton's method thus offers very fast convergence when compared to fixed point iteration.

12.3 Take-Away Messages

:

- Quadratic convergence rate.
- We can achieve machine precision.
- Most expensive part of this operation is solving a linear system, which is an $O(n^3)$ operation.
- It's possible to improve upon the time-complexity by taking advantage of the structure of the problem. (We'll touch on this later!)

13 Minimization

Consider finding a value x for which the function $f(x) : \mathbb{R}^n \longrightarrow \mathbb{R}$ attains its minimum value. If f is smooth, $\frac{\partial f}{\partial x}\big|_{x^*} = 0$ at local minima. This means we can go about minimizing f by applying Newton root finding to $\frac{\partial f}{\partial x} = 0$. Consider:

$$\nabla f(x + \Delta x) \approx \nabla f(x) + \nabla^2 f(x)\Delta x = 0 \implies \Delta x = -(\nabla^2 f(x))^{-1}\nabla f(x) = 0 \tag{50}$$

This means we can find our minimum by repeatedly updating our estimate x as $x \leftarrow x + \Delta x$ until convergence. The intuition for this is that we are fitting a quadratic approximation to $f(x)$, using a Taylor expansion at the current guess of the solution. We can exactly minimize quadratic approximations.
Remember to look at the Jupyter notebook in the lecture video for an example $f(x) = x^4 + x^3 - x^2 - x$.

13.1 Take-Away Messages

- Newton's root finding is a **local** method. It will find the **closest** fixed point to the initial guess, which could be a maxima, minima, or a saddle point.

13.2 Sufficient Conditions

In the example, we saw that Newton's method doesn't really care about whether it is maximizing or minimizing the function at hand. So how do we know which we're doing? Let's think about the scalar case:

$$\Delta x = -(\nabla^2 f)^{-1}\nabla f \tag{51}$$

Here, the $-$ signifies performing descent on the function value, the term $(\nabla^2 f)^{-1}$ is analogous to a learning rate, and ∇f is simply the gradient.

$$\nabla^2 f > 0 \implies \text{Descent (minimization)} \tag{52}$$
$$\nabla^2 f < 0 \implies \text{Ascent (maximization)} \tag{53}$$

In the $\mathbb{R}^n$ case, the equivalent conditions are whether the second derivative Hessians are positive definite (for descent / minimization) or vice versa.
If $\nabla^2 f > 0$ everywhere in the function domain, we have that f is strongly convex. We can always find a global minimum by Newton's method. Unfortunately, this doesn't really hold true for hard / nonlinear problems.

13.3 Regularization

What do we do if we don't have a strongly convex function? The practical solution is to make sure we are actually always minimizing while executing Newton's method.

$$H \leftarrow \nabla^2 f \tag{54}$$

Now while $H \nsucc 0$ (while the Hessian is not positive definite), we can iteratively set $H \leftarrow H + \beta I$, where β is a scalar hyper-parameter.

Algorithm 2 Regularization / Damped Newton's Method

```
1: $H \leftarrow \nabla^2 f$
2: while $H \nsucc 0$ do                         ▷ Not positive definite.
3:     $H \leftarrow H + \beta I$                 ▷ $\beta$ is a scalar hyperparameter.
4: $\Delta x = -H^{-1}\nabla f$
5: $x \leftarrow x + \Delta x$
```

After modifying the Hessian in this way, we can then take a "Newton" step with $\Delta x = -H^{-1}\nabla f$, $x \leftarrow x + \Delta x$. This modified Newton's method is called "Damped Newton's Method". The trick of modifying the Hessian is called regularization. It guarantees we are performing descent (minimization), and shrinks the step size of the step taken. Remember to check out the lecture video for the behavior in our example!

13.4 Line Search

One problem that we ran into, is that the step Δx is often too big, and overshoots the minimum of the function. To fix this, we can check the function value at $f(x + \Delta x)$ and "backtrack" until we get a "good" reduction.
There are many strategies that exist to do this. One such strategy that is both simple and effective, is the Armijo rule:

Algorithm 3 Armijo Rule

```
1: $\alpha = 1$                                   ▷ Step Length
2: while $f(x + \alpha\Delta x) > f(x) + b\alpha\nabla f(x)^T\Delta x$ do
3:     ▷ $\alpha\nabla f(x)^T\Delta x$ is the expected reduction from gradient, and $b$ is the tolerance.
4:     $\alpha \leftarrow c\alpha$                 ▷ $c$ is a scalar $< 1$.
```

The intuition for this is to make sure the step size agrees with linearization within some tolerance b. Typical values of the parameters above are - $c = 0.5$, $b = 10^{-4}$ to 0.1.

13.5 Take-Away Messages

- Newton's method with simple and cheap modifications or globalization strategies (such as regularization), is extremely effective at finding local minima.

14 References

Remember to check out [1] for more details on this.

References

[1] J. Nocedal et al. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2006. ISBN: 9780387303031. URL: `https://books.google.com/books?id=eNlPAAAAMAAJ`.

[2] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2005. ISBN: 9780471649908. URL: `https://books.google.com/books?id=AOOXDwAAQBAJ`.

[3] S.H. Strogatz and M. Dichter. *Nonlinear Dynamics and Chaos, 2nd ed. SET with Student Solutions Manual*. Studies in Nonlinearity. Avalon Publishing, 2016. ISBN: 9780813350844. URL: `https://books.google.com/books?id=vUWhDAEACAAJ`.

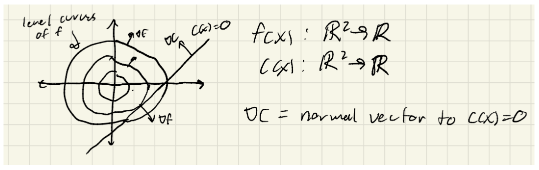
level curves of f
∇f
∇c
$c(x)=0$
$f(x) : \mathbb{R}^2 \to \mathbb{R}$
$c(x) : \mathbb{R}^2 \to \mathbb{R}$
∇f
∇c = normal vector to $c(x)=0$

Figure 5: Level curves of $f(x)$

15 LECTURE 4

16 Agenda

This lecture covers constrained minimization.

17 Equality Constraints

Let's consider the following minimization problem, subject to some *equality constraints*:

$$\min_x f(x) \quad ; \quad f(x) : \mathbb{R}^n \longrightarrow \mathbb{R} \tag{55}$$

$$\ni c(x) = 0 \quad ; \quad c(x) : \mathbb{R}^n \longrightarrow \mathbb{R}^m \tag{56}$$

17.1 First order necessary conditions

The first order necessary conditions for the x at which the function attains its minimum:

- Need $\nabla f(x) = 0$ in the unconstrained / free directions.
- Need $c(x) = 0$

Thus any non-zero component of ∇f must be normal to the constraint surface / manifold.

$$\nabla f + \lambda \nabla c = 0 \text{ for some } \lambda \in \mathbb{R} \tag{57}$$

where λ is a Lagrange multiplier or "Dual variable". In general, we have that:

$$\frac{\partial f}{\partial x} + \lambda^T \frac{\partial c}{\partial x} = 0 \tag{58}$$

Based on this gradient condition, we define:

$$L(x, \lambda) = f(x) + \lambda^T c(x) \tag{59}$$

where $L(x, \lambda)$ represents the Lagrangian. This is subject to:

$$\nabla_x L(x, \lambda) = \nabla f + (\frac{\partial c}{\partial x})^T \lambda = 0 \tag{60}$$

$$\nabla_\lambda L(x, \lambda) = c(x) \tag{61}$$

Together, these conditions represent the KKT conditions. We can now solve this jointly in x and λ as a root finding problem, using Newton's method.

$$\nabla_x L(x + \Delta x, \lambda + \Delta\lambda) \approx \nabla_x L(x, \lambda) + \frac{\partial^2 L}{\partial x^2}\Delta x + \frac{\partial^2 L}{\partial x \partial \lambda}\Delta\lambda \tag{62}$$

$$\nabla_\lambda L(x + \Delta x, \lambda) \approx c(x) + \frac{\partial c}{\partial x}\Delta x \tag{63}$$

where $\frac{\partial^2 L}{\partial x \partial \lambda} = (\frac{\partial c}{\partial x})^T$. This can be written as the following matrix system:

$$\begin{bmatrix} \frac{\partial^2 L}{\partial x^2} & (\frac{\partial c}{\partial x})^T \\ \frac{\partial c}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -\nabla_x L(x, \lambda) \\ -c(x) \end{bmatrix} \tag{64}$$

The first matrix here is the Hessian of the Lagrangian, and describes the KKT system.

17.2 Gauss-Newton Method

$$\frac{\partial^2 L}{\partial x^2} = \nabla^2 f + \frac{\partial}{\partial x}\left[(\frac{\partial c}{\partial x})^T \lambda\right] \tag{65}$$

Here, $\frac{\partial}{\partial x}\left[(\frac{\partial c}{\partial x})^T \lambda\right]$ is expensive to compute. We often drop this term, ignoring the "constraint curvature". Dropping this term is called Gauss-Newton method. This is equivalent to linearizing the system first, and then performing Newton's method.
Check out the example from the Lecture on this.

17.3 Take-Away Message

- May need to regularize $\frac{\partial^2 L}{\partial x^2}$ in Newton's method, even if $\nabla^2 f = 0$.
- Gauss-Newton is often used in practice because it converges almost as fast, and is cheaper per iteration.

18 Inequality constraints

More generally, we have to reconcile with inequality constraints as well. Consider a minimization problem subject to inequality constraint:

$$\min_x f(x) \tag{66}$$

$$\ni c(x) \geq 0 \tag{67}$$

Let's look at the case of purely inequality constraints for now. In general, these methods are combined with the equality constraint methods above, to handle general inequality / equality constraints in the same problem.

18.1 First-Order Necessary Conditions

As before, the first order conditions specify that -

- Need $\nabla f(x) = 0$ in the unconstrained / free directions.
- Need $c(x) \geq 0$ Note the inequality here.

Mathematically,

$$\nabla f - (\frac{\partial c}{\partial x})^T \lambda = 0 \leftarrow \text{"Stationarity"} \tag{68}$$

$$c(x) \geq 0 \leftarrow \text{"Primal Feasibility"} \lambda \quad \geq 0 \leftarrow \text{"Dual Feasibility"} \lambda^T c(x) = 0 \leftarrow \text{"Complementarity"} \tag{69}$$

Together, these conditions specify the full set of KKT conditions.

18.2 Intuition

The intuition for these conditions is as follows:

- If constraint is active, (we are on the constraint manifold), we have $c(x) = 0 \implies \lambda = 0$. This is the same as the equality case.
- If the constraint is inactive, we have $c(x) > 0 \implies \lambda = 0$, which is the same as the *unconstrained* case.
- Essentially complementarity ensures either one of λ and $c(x)$ will be 0, or "on /off switching" of constraints.

18.3 Algorithms

The implementation of optimization methods here is much hard than the equality case; we can't directly apply Newton's method to the KKT conditions. There are many options we can use, with different trade offs.

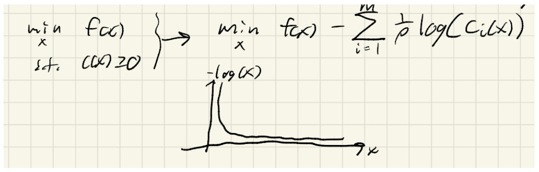

Figure 6: Barrier Function

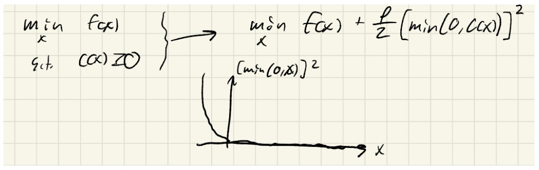

Figure 7: Penalty Method Objective

18.4 Active-Set Method

- Applies when you have a good way of knowing which constraints are active / inactive.
- Solves equality constrained problem (when it is active).
- Very fast if you have a good heuristic.
- Can be very (combinatorially) bad if you don't know which constraints are active.

18.5 Barrier / Interior-Point Method

- We replace the inequalities with a "barrier function", in the objective, that blows up at the constraint boundary fig. 6.
- This is the gold standard for small to medium *convex* problems.
- Requires a lot of hacks and tricks to get working for non-convex problems.

18.6 Penalty Method

- Replace the inequality constraint with an objective term that penalizes violations fig. 7.
- It's easy to implement.
- Has issues with being ill-conditioned (the Hessian has a huge spread of eigenvalues, that causes Newton's method to struggle).
- Difficult to achieve high accuracy.

18.7 Augmented Lagrangian

- Add a Lagrange multiplier estimate to the penalty method, to fix issues that penalty method faced.

$$\min_x f(x) - \tilde{\lambda}^T c(x) + \rho/2\big[\min(0, c(x))\big]^2 \tag{70}$$

where $f(x) - \tilde{\lambda}^T c(x) + \rho/2\big[min(0, c(x))\big]^2 = L_\rho(x, \lambda)$ is the augmented lagrangian. Remember, here we want to penalize the constraints being violated, that's why we have a $-\tilde{\lambda}$ instead of $+\lambda$. We first minimize with respect to x (with a fixed $\tilde{\lambda}$, and then we update $\tilde{\lambda}$ by "offloading" penalty term at each iteration:

$$\frac{\partial f}{\partial x} - \tilde{\lambda}\frac{\partial c}{\partial x} + \rho c(x)^T \frac{\partial c}{\partial x} = \frac{\partial f}{\partial x} - \big[\tilde{\lambda} - \rho c(x)\big]^T \frac{\partial c}{\partial x} = 0 \implies \tilde{\lambda} \leftarrow \tilde{\lambda} - \rho c(x) \text{ for active constraints.} \tag{71}$$

Algorithmically, this can be expressed as:

Algorithm 4 Augmented Lagrangian method

```
1: while Not Converged do
2:     $\min_x L_\rho(x, \tilde{\lambda}$                          ▷ Minimize w.r.t $x$.
3:     $\tilde{\lambda} \leftarrow \tilde{\lambda} - \max(0, \tilde{\lambda} - \rho c(x))$   ▷ Update the multipliers, clamping to guarantee non-negativity.
4:     $\rho \leftarrow \alpha\rho$                                ▷ Increase penalty. Typically, $\alpha \approx 10$
```

- This fixes the ill-conditioning that penalty method suffers.
- It converges fast (super-linearly) to moderate precision.
- Works well on non-convex problems too!

18.8 Quadratic Program Example!

Consider the problem:

$$\min_x \ \frac{1}{2}x^T Qx + q^T x \quad \ni Q > 0 \text{ (Convex)} \tag{72}$$

$$\ni Ax \leq b \tag{73}$$

$$\ni Cx = d \tag{74}$$

Here have a quadratic objective, and linear constraints. This type of problem is very common and useful in control, and can be solved very fast online (in the order of KHz). Remember to check out the example in the lecture video!

18.9 Additional Notes

- In general, we still need regularization and line searches in the constrained setting.
- Linear searches get a little more complicated.

19 References

Remember to check out [1] for more on constrained minimization!

References

[1] J. Nocedal et al. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2006. ISBN: 9780387303031. URL: `https://books.google.com/books?id=eNlPAAAAMAAJ`.

[2] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2005. ISBN: 9780471649908. URL: `https://books.google.com/books?id=AOOXDwAAQBAJ`.

[3] S.H. Strogatz and M. Dichter. *Nonlinear Dynamics and Chaos, 2nd ed. SET with Student Solutions Manual*. Studies in Nonlinearity. Avalon Publishing, 2016. ISBN: 9780813350844. URL: `https://books.google.com/books?id=vUWhDAEACAAJ`.

20 LECTURE 5

21 Agenda

In this lecture, we cover regularization, and line searches with constraints. We'll then move on to deterministic optimal control.

22 Regularization and Duality

Consider the following problem:

$$\min_x f(x) \tag{75}$$

$$\ni c(x) = 0 \tag{76}$$

We can express this as:

$$\min_x f(x) + P_\infty(c(x)) \tag{77}$$

$$P_\infty(x) = \begin{cases} 0, & x = 0 \\ +\infty, & x \neq 0 \end{cases} \tag{78}$$

Practically, this is terrible, but we can get the same effect by solving:

$$\min_x \max_\lambda f(x) + \lambda^T c(x) \tag{79}$$

Whenever $c(x) \neq 0$, the inner problem gives $+\infty$. Similarly for inequalities:

$$\min_x \; f(x) \tag{80}$$

$$\ni c(x) \geq 0 \tag{81}$$

$$\implies \tag{82}$$

$$\min_x \; f(x) + P_\infty^+(c(x)) \tag{83}$$

Here, we have:

$$P_\infty^+(x) = \begin{cases} 0, & x \geq 0 \\ +\infty, & x < 0 \end{cases} \implies \tag{84}$$

$$\min_x \max_{\lambda \geq 0} f(x) - \lambda c(x) \tag{85}$$

where $f(x) - \lambda c(x)$ is our Lagrangian $L(x, \lambda)$.

For convex problems, one can switch the order of the min and max, and the solution does not change; this is the notion of "dual problems"! This isn't true in general, however, i.e. for non-convex problems.

The interpretation of this is that the KKT conditions define a saddle point in the space of (x, λ).

The KKT system should have $\dim(x)$ positive eigenvalues and $\dim(\lambda)$ negative eigenvalues at an optimum. Such a system is known as a "Quasi-definite" linear system.

22.1 Take-Away Messages:

When regularizing a KKT system, the lower-right block should be negative!

$$\begin{bmatrix} H + \alpha I & C^T \\ C & -\alpha I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -\nabla_x L \\ -c(x) \end{bmatrix}, \alpha > 0 \tag{86}$$

This makes the system a quasi-definite one.
Remember to check out the example of this in the lecture video. We observe that we still overshoot the solution, so we need a line search!

22.2 Merit Functions

How do we a do a line search on a root-finding problem? Consider:

$$\text{find} x* \ni \; c(x*) = 0 \tag{87}$$

First, we define a scalar "merit function" $P(x)$, that measure distance from a solution. A few standard choices for this merit function include:

$$P(x) = \frac{1}{2}c(x)^T c(x) = \frac{1}{2}||c(x)||_2^2 \tag{88}$$

$$\text{or} \quad P(x) = ||c(x)||_1 \quad \text{Note: Could use any norm.} \tag{89}$$

Now we could just do Armijo on $P(x)$:

Algorithm 5 Armijo Rule on $P(x)$

```
1: $\alpha = 1$                                                   ▷ Step Length
2: while $f(x + \alpha\Delta x) > f(x) + b\alpha\nabla f(x)^T\Delta x$ do
3:         ▷ $\alpha\nabla f(x)^T\Delta x$ is the expected reduction from gradient, and $b$ is the tolerance.
4:     $\alpha \leftarrow c\alpha$                                    ▷ $c$ is a scalar $< 1$.
5: $x \leftarrow \alpha\Delta x$
```

22.3 Constrained Minimization

How about constrained minimization? Here, we want to come up with an option that specifies how much we are violating the constraint, as well as how far off the optimum we are / minimizing the objective funciton. Consider the problem:

$$\min_x f(x) \tag{90}$$
$$\ni \; c(x) \geq 0 \tag{91}$$
$$\ni \; d(x) = 0 \tag{92}$$
$$\Longrightarrow \tag{93}$$
$$L(x, \lambda, \mu) = f(x) - \lambda^T c(x) + \mu^T d(x) \tag{94}$$

We have lots of options for merit functions. One option is:

$$P(x, \lambda, \mu) = \frac{1}{2}||\nabla L(x, \lambda, \mu)||_2^2 \tag{95}$$
$$\tag{96}$$

Here, the term $\nabla L(x, \lambda, \mu)$ is the KKT residual:

$$\begin{bmatrix} \nabla_x L(x, \lambda, \mu) \\ \min(0, c(x)) \\ d(x) \end{bmatrix} \tag{97}$$

However, this isn't the best option to use, because evaluating the gradient of the KKT condition is as expensive as the newton step solve itself.
Another option is:

$$P(x, \lambda, \mu) = f(x) + \rho \; || \begin{bmatrix} \min(0, c(x)) \\ d(x) \end{bmatrix} ||_1 \tag{98}$$

Here, ρ is a scalar trade off between the objective minimization and constraint satisfaction. Also remember that any norm works here (in place of the 1 norm depicted), but using the 1 norm is the most common. This option gives us flexibility, because we can pick trade off ρ - initially we can set this to be low, to drive us close to the optimum solution, and when we are close to the optimum, we can increase ρ to ensure we satisfy the constraints.

Yet another option is:

$$P(x, \lambda, \mu) = f(x) - \tilde{\lambda}^T c(x) + \tilde{\mu}^T d(x) + \frac{\rho}{2} || \min(0, c(x)||_2^2 + \frac{\rho}{2} ||d(x)||_2^2 \tag{99}$$

which is the augmented Lagrangian itself.
Remember to check out the example in the lecture video!

22.4 Take-Away Messages (from the example)

- $P(x)$ based on the KKT residual is expensive.
- Excessively large penalty weights can cause problems.
- Augmented Lagrangian methods come with a merit function for free. So if we're using the Augmented Lagrangian to solve the problem, just use this as a merit function.

22.5 Deterministic Optimal Control

Let's consider the following control problem:

$$\min_{x(t), u(t)} = J(x(t), u(t)) = \int_{t_0}^{t_f} L(x(t), u(t)) dt + L_F(x(t_f)) \tag{100}$$

$$\ni \dot{x}(t) = f(x(t), u(t)) \tag{101}$$

$$\ni \text{Any other constraints} \tag{102}$$

Here, we minimize across "state" or "input" trajectories. J represents our cost function, $L(x(t), u(t))$ represents our "stage cost", $L_F(x(t_f))$ represents a "terminal cost", $\dot{x}(t) = f(x(t), u(t))$ are the dynamics constraint.

This is an "infinite dimensional" problem in the sense that an infinite amount of discrete time control points required to fully specify the control to be applied.

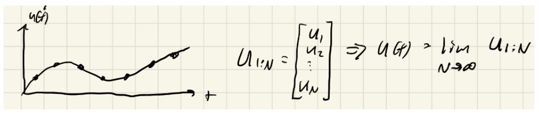
$u(t)$
t
$u_{1:N} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix} \Rightarrow u(t) = \lim_{N\to\infty} u_{1:N}$

Figure 8: Deterministic optimal control problem

- The solutions to this problem are open loop trajectories.
- Now, there are a few control problems with analytic solutions in continuous time, but not many.
- We focus on the discrete-time setting, where we have tractable algorithms.

22.6 Discrete Time

Consider a discrete-time version of this problem.

$$\min_{x_{1:N}, u_{1:N-1}} J(x_{1:N}, u_{1:N-1}) = \sum_{k=1}^{N-1} L(x_k, u_k) + L_F(x_N) \tag{103}$$

$$\ni x_{n+1} = f(x_n, u_n) \tag{104}$$

$$\ni u_{\min} \leq u_k \leq u_{\max} \text{ Torque limits.} \tag{105}$$

$$\ni c(x_k) \leq 0 \ \forall k \text{ Obstacle / collision constraints.} \tag{106}$$

- This version of the problem is now a finite dimensional problem.
- Samples x_k, u_k are often called “knot points”.
- We can convert continuous systems to discrete-time problems using integration methods such as the Runge-Kutta method etc.
- Finally, we can convert back from discrete-time problems to continuous problems using interpolation.

23 LECTURE 6

24 Agenda

This lecture covers Pontryagin's Minimum Principle, and an introduction to Linear Quadratic Regulators.

25 Pontryagin's Minimum Principle

- The Pontryagin's Minimum Principle is also a "maximum principle" if instead of minimizing a cost function, we are maximizing a reward function.
- Essentially provides first order necessary conditions of deterministic optimal control problems.
- In discrete time, it's just a special case of the KKT.

Consider the problem we have before:

$$\min_{x_{1:N}, u_{1:N-1}} J(x_{1:N}, u_{1:N-1}) = \sum_{k=1}^{N-1} l(x_k, u_k) + l_F(x_N) \tag{107}$$

$$\ni x_{n+1} = f(x_n, u_n) \tag{108}$$

In this setting, we will consider torque limits applied to the problem, but it's hard to handle constraints on state (such as collision constraints, like we had before).
We can form the Lagrangian of this problem as follows:

$$L = \sum_{k=1}^{N-1} \Big[l(x_k, u_k) + \lambda_{k+1}^{+} (f(x_k, u_k) - x_{k+1}) \Big] + l_F(x_N) \tag{109}$$

$$\tag{110}$$

This result is usually stated in terms of the "Hamiltonian":

$$H(x, u, \lambda) = l(x, u) + \lambda^T f(x, u) \tag{111}$$

Plugging in H into the Lagrangian L:

$$L = H(x_1, u_1, \lambda_2) + \Big[\sum_{k=2}^{N-1} H(x_k, u_k, \lambda_{k+1}) - \lambda_k^T x_k \Big] + l_F(x_N) - \lambda_N^T x_N \tag{112}$$

Note the change in indexing of the summation. If we take derivatives with respect to x and λ:

$$\frac{\partial L}{\partial \lambda_k} = \frac{\partial H}{\partial \lambda_k} - x_{k+1} = f(x_k, u_k) - x_{k+1} = 0 \tag{113}$$

$$\frac{\partial L}{\partial x_k} = \frac{\partial H}{\partial x_k} - \lambda_k^T = \frac{\partial l}{\partial x_k} + \lambda_{k+1}^T \frac{\partial f}{\partial x_k} - \lambda_k^T = 0 \tag{114}$$

$$\frac{\partial L}{\partial x_N} = \frac{\partial l_F}{\partial x_N} - \lambda_N^T = 0 \tag{115}$$

For u, we write the min explicitly to handle the torque limits:

$$u_k = \arg\min_{\tilde{u}} H(x_k, \tilde{u}, \lambda_{k+1}) \tag{116}$$

$$\ni \tilde{u} \in \mathcal{U} \tag{117}$$

Here, $\tilde{u} \in \mathcal{U}$ is shorthand for "in feasible set", for example, $u_{\min} \leq \tilde{u} \leq u_{\max}$.

In summary, we have:

$$\begin{aligned}
x_{k+1} &= \nabla_\lambda H(x_k, u_k, \lambda_{k+1}) = f(x_k, u_k) && (118)\\
\lambda_k &= \nabla_x H(x_k, u_k, \lambda_{k+1}) = \nabla_x l(x_k, u_k) + (\frac{\partial f}{\partial x})^T \lambda_{k+1} && (119)\\
u_k &= \arg\min_{\tilde{u}} H(x_k, \tilde{u}, \lambda_{k+1}) && (120)\\
&\ni \ \tilde{u} \in \mathcal{U} && (121)\\
\lambda_N &= \frac{\partial l_F}{\partial X_N} && (122)
\end{aligned}$$

Now these can be stated almost identically in continuous time:

$$\begin{aligned}
\dot{x} &= \nabla_\lambda H(x, u, \lambda) = f_{\text{continuous}}(x, u) && (123)\\
\dot{\lambda} &= \nabla_x H(x, u, \lambda) = \nabla_x l(x, u) + (\frac{\partial f_{\text{continuous}}}{\partial x})^T \lambda && (124)\\
u &= \arg\min_{\tilde{u}} H(x, \tilde{u}, \lambda) && (125)\\
&\ni \ \tilde{u} \in \mathcal{U} && (126)\\
\lambda_N &= \frac{\partial l_F}{\partial X_N} && (127)
\end{aligned}$$

25.1 Notes

- Historically, many algorithms were based on forward / backward integration of the continuous ODEs for $x(t), \lambda(t)$ for performing gradient descent on $u(t)$.
- Thse are called "indirect" and / or "shooting" methods.
- In continuous time, $\lambda(t)$ is called "co-state" trajectory.
- These methods have largely fallen out of favor as computers and solvers have improved.

26 Linear Quadratic Regulator (LQR)

A very common class of controllers is the Linear Quadratic Regulator. In this setting, we have a quadratic cost, and linear dynamics, specified as:

$$\begin{aligned}
&\min_{x_{1:N}, u_{1:N-1}} \sum_{k=1}^{N-1} \left[\frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R_k u_k\right] + \frac{1}{2} x_N^T Q_N x_N && (128)\\
&\ni \ x_{k+1} = A_k x_k + B_k u_k && (129)
\end{aligned}$$

Here, we have: $Q \geq 0, R \geq 0$.

- The goal of this problem is to drive the system to the origin.
- It's considered a "time-invariant" LQR if $A_k = A, B_k = B, Q_k = Q, R_k = R \ \forall \ k$, and is a "time-varying" LQR (TVLQR) otherwise.
- We typically use time-invariant LQRs for stabilizing an equilibrium, and TVLQR for tracking trajectories.
- Can (locally) approximate many non-linear problems, and are thus very commonly used.
- There are also many extensions of LQR, including the infinite horizon case, and stochastic LQR.
- It's been called the "crown jewel of control theory."

26.1 LQR with Indirect Shooting

Consider:

$$x_{k+1} = Ax_k + Bu_k \tag{130}$$

$$\lambda_k = Qx_k + A^T\lambda_{k+1},\ \lambda_N = Qx_N \tag{131}$$

$$u_k = -R^{-1}B^T\lambda_{k+1} \text{ This gives the gradients of the cost w.r.t.} u_k \tag{132}$$

The procedure for LQR with indirect shooting is:

1. Start with an initial guess trajectory.
2. Simulate (or "rollout") to get $x(t)$.
3. Backward pass to get $\lambda(t)$ and $\Delta u(t)$.
4. Rollout with line search on Δu.
5. Go to (3) until convergence.

26.2 Example

Check out the example of the double integrator in the lecture. Here, we have -

$$\dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{133}$$

Think of this as a "sliding brick" on ice, without friction. Here, the discrete-time version of this system is -

$$x_{k+1} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q_k \\ \dot{q_k} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}h^2 \\ h \end{bmatrix} u_k \tag{134}$$

where $\begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}$ represents the A matrix, and $\begin{bmatrix} \frac{1}{2}h^2 \\ h \end{bmatrix}$ represents the B matrix.

26.3 Bryson's Rule

- Cost-tuning heuristic.
- Set diagonal elements of Q and R to $\frac{1}{\text{max value}^2}$.
- Normalizes all cost terms to 1.
- Good starting point for tuning.
- In the scalar case, if we have:

$$\frac{1}{2}Qx^2 + \frac{1}{2}Ru^2 \tag{135}$$

Then we can set Q such that $Qx^2 = \frac{1}{x^2_{\text{max}}}x^2_{\text{max}} = 1$, and similarly R such that: $Ru^2 = \frac{1}{u^2_{\text{max}}}u^2_{\text{max}} = 1$

27 LECTURE 7

27.1 Agenda

- LQR as a QP
- Riccati Recursion

27.2 The LQR Problem

Consider the LQR problem -

$$\min_{x_{1:n},u_{1:n-1}} J = \Big[\sum_{n=1}^{N-1} \frac{1}{2}x_n^T Q_n x_n + \frac{1}{2}u_n^T R_n u_n\Big] + \frac{1}{2}x_N^T Q_N x_N \tag{136}$$

$$\ni \; x_{n+1} = A_n x_n + B_n u_n \tag{137}$$

Remember, we have that $Q \geq 0$ and $R > 0$.

27.2.1 Framing LQR as a QP

Let us assume the initial state x_1 is given (and is not a decision variable). Define z:

$$z = \begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ . \\ . \\ x_N \end{bmatrix} \tag{138}$$

Also define H:

$$H = \begin{bmatrix} R_1 & 0 & ... & 0 \\ 0 & Q_2 & ... & 0 \\ & & . & \\ 0 & 0 & ... & Q_N \end{bmatrix} \tag{139}$$

such that $J = \frac{1}{2}z^T H z$. We also define C and d:

$$C = \begin{bmatrix} B_1 & (-I) & ... & ... & ... & 0 \\ 0 & A & B & (-I) & ... & 0 \\ & & . & & & \\ 0 & 0 & ... & A_{N-1} & B_{N-1} & (-I) \end{bmatrix} \tag{140}$$

$$d = \begin{bmatrix} -A_1 x_1 \\ 0 \\ . \\ 0 \end{bmatrix} \tag{141}$$

such that $Cz = d$. We now have the LQR problem can be written as a standard QP:

$$\min_z \frac{1}{2} z^T H z \tag{142}$$

$$\ni Cz = d \tag{143}$$

The Lagrangian of this QP is:

$$L(z, \lambda) = \frac{1}{2} z^T H z + \lambda^T \big[Cz - d\big] \tag{144}$$

and the KKT conditions are:

$$\nabla_z L = Hz + C^T \lambda = 0 \tag{145}$$

$$\nabla_\lambda L = Cz - d = 0 \tag{146}$$

This may be written as:

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix} \tag{147}$$

We get the exact answer by solving this one linear system!

27.2.2 Example

Remember to check out the example on the double integrator / sliding brick on ice. We can compare shooting to the QP solution of the LQR problem.

27.2.3 A closer look at the LQR QP

The QP KKT system is very sparse (i.e. lots of zeros in the constituent matrices), and has a lot of structure:

$$\left[\begin{array}{cccccccccc} R & & & & & & . & B^T & & \\ & Q & & & & & . & -I & A^T & \\ & & R & & & & . & & B^T & \\ & & & Q & & & . & & -I & A^T \\ & & & & R & & . & & & B^T \\ & & & & & Q_N & . & & & -I \\ . & . & . & . & . & . & . & . & . & . \\ B & -I & & & & & . & 0 & 0 & 0 \\ & A & B & -I & & & . & 0 & 0 & 0 \\ & & & A & B & -I & . & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ x_3 \\ u_3 \\ x_4 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -Ax_1 \\ 0 \\ 0 \end{bmatrix} \tag{148}$$

- Consider the last line of upper block of this system (i.e. above the dotted line):

$$Q_N x_4 - \lambda_4 = 0 \tag{149}$$

$$\implies \lambda_4 = Q_N x_4 \tag{150}$$

Now consider the second last line of the upper block of this system:

$$Ru_3 + B^T \lambda_4 = Ru_3 + B^T Q_N x_4 = 0 \tag{151}$$

Plugging the dynamics into this, we have:

$$Ru_3 + B^T Q_N (Ax_3 + Bu_3) = 0 \tag{152}$$

$$\implies u_3 = -(R + B^T Q_N B)^{-1} B^T Q_N A x_3 \tag{153}$$

We can define $K_3 = (R + B^T Q_N B)^{-1} B^T Q_N A$ in the above equation.
Finally, consider the third line of this system.

$$Qx_3 - \lambda_3 + A^T \lambda_4 = 0 \tag{154}$$

We can then manipulate this by first substituting λ_4, then plugging in the dynamics for x_4, and finally using the feedback law:

$$Qx_3 - \lambda_3 + A^T Q_N x_4 = 0 \tag{155}$$

$$\implies Qx_3 - \lambda_3 + A^T Q_N (Ax_3 + Bu_3) = 0 \tag{156}$$

$$\implies Qx_3 - \lambda_3 + A^T Q_N (A - BK) x_3 = 0 \tag{157}$$

$$\implies \lambda_3 = (Q + A^T Q_N (A - BK)) x_3 \tag{158}$$

We can then define $P_3 = (Q + A^T Q_N (A - BK))$.
We now have a recursion for K and P:

$$P_N = Q_N \tag{159}$$

$$K_n = (R + B^T P_{n+1} B)^{-1} B^T P_{n+1} A \tag{160}$$

$$P_n = Q + A^T P_{n+1} (A - BK_n) \tag{161}$$

- This is called a Riccati equation / recursion.
- We can solve the QP by doing a backward Riccati recursion followed by a forward rollout starting from x_1.
- This has complexity $\mathcal{O}(N(n+m)^3$ instead of $\mathcal{O}(N^3(n+m)^3$ for the naive QP solution. This carries over to the general non-linear case.
- Even more importantly, we now have a feedback policy instead of an open loop trajectory to execute.

27.2.4 Example

Remember to check out the lecture video for an example on LQR with Riccati open-loop, and closed loop with noise.

27.2.5 Infinite Horizon

Let's now consider the infinite horizon case.

- For the time-invariant LQR, K matrices converge to constant values over the infinite horizon.
- For stabilization problems we almost always use the constant K.
- We can solve for this explicitly in Julia / Matlab.

27.3 Controllability

How do we know if LQR will work? For the time-invariant case, there is a simple answer. For any initial state x_0, x_n is given by:

$$\begin{aligned} x_n &= Ax_{n-1} + Bu_{n-1} && (162)\\ &= A(Ax_{n-2} + Bu_{n-2}) + Bu_{n-1} && (163)\\ &= A^n x_0 + A_{n-1}Bu_0 + ... + Bu_{n-1} && (164)\\ &= \begin{bmatrix} B & AB & A^2B & ... & A^{n-1}B \end{bmatrix} \begin{bmatrix} u_{n-1} \\ u_{n-2} \\ . \\ . \\ u_0 \end{bmatrix} + A^n x_0 && (165) \end{aligned}$$

Here, we may define a controllability matrix C as:

$$C = \begin{bmatrix} B & AB & A^2B & ... & A^{n-1}B \end{bmatrix} \tag{166}$$

In order to drive any x_0 to any desired x_n, the controllability matrix must have full row rank:

$$\begin{aligned} rank(C) &= n && (167)\\ \ni n &= dim(x) && (168) \end{aligned}$$

This is equivalent to solving the following least-squares problem for $u_{0:n-1}$:

$$\begin{bmatrix} u_{n-1} \\ u_{n-2} \\ . \\ . \\ u_0 \end{bmatrix} = \begin{bmatrix} C^T(CC^T)^{-1} \end{bmatrix} \begin{bmatrix} (x_n - A^n x_0) \end{bmatrix} \tag{169}$$

Here, $\begin{bmatrix} C^T(CC^T)^{-1} \end{bmatrix}$ is the pseudo-inverse of controllability C, and requires CC^T to be invertible. We can stop at n time steps because the Cayley-Hamilton theorem says that A^n can e written as a linear combination of smaller powers of A:

$$A^n = \sum_{k=0}^{n-1} \alpha_k A^k \tag{170}$$

for some α_k. Therefore adding more timesteps / columns to C can't increase the rank of C.

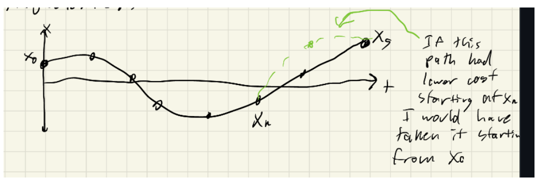
x
x0
Xg
t
Xn
If this path had lower cost starting at Xn I would have taken it startin from Xo

Figure 9: Bellman's Principle

28 LECTURE 8

28.1 Agenda

- Dynamic Programming
- Convexity

28.2 Bellman's Principle

- Optimal control problems have an inherently sequential structure.
- Past control inputs affect future states but future control inputs cannot affect past states.
- Bellman's principle (i.e. The principle of optimality), states the cocnsequences of this for optimal trajectories.
- Sub-trajectories of optimal trajectories have to be optimal for the appropriately defined sub-problem.

28.3 Dynamic Programming

- Bellman's Principle suggests starting from the end of the trajectory and working backwards.
- We've already seen hints of this in the Riccati equation, and in the co-state / multiplier equation from Pontryagin.
- Define "optimal cost-to-go" a.k.a "value function" $V_N(x)$.
- Encodes the cost incurred starting from state x at time k if we act optimally.
- For the LQR setting, we have:

$$V_N(x) = \frac{1}{2}x^T Q_N x = \frac{1}{2}x^T P_N x \tag{171}$$

- Now we can back up one time step, and compute $V_{N-1}(x)$:

$$\min_u \frac{1}{2}x_{N-1}^T Q x_{N-1} + \frac{1}{2}u^T R u + V_N(A_{N-1}x_{N-1} + B_{N-1}u) \tag{172}$$

$$= \min_u \frac{1}{2}u^T R u + \frac{1}{2}(Ax_{N-1} + B_{N-1}u)^T Q_N (Ax_{N-1} + B_{N-1}u) \tag{173}$$

Since for the optimal action we have ∇_u of this cost $= 0$, we have:

$$u^T R_{N-1} + (Ax_{N-1} + B_{N-1}u)^T Q B_{N-1} = 0 \tag{174}$$

$$\implies u_{N-1} = -(R_{N-1} + B_{N-1}^T Q_N B_{N-1})^{-1} B_{N-1}^T Q_N A_{N-1} x_{N-1} \tag{175}$$

We may define K_{N-1} as $(R_{N-1} + B_{N-1}^T Q_N B_{N-1})^{-1} B_{N-1}^T Q_N A_{N-1}$ for convenience.

- Plugging in $u = -Kx$ into our expression for $V_{N-1}(x)$, we have:

$$V_{N-1}(x) = \frac{1}{2}x^T(Q_{N-1} + K_{N-1}^T R_{N-1}K + (A_{N-1} - B_{N-1}K_{N-1})^T Q_N(A_{N-1} - B_{N-1}K_{N-1}))x \tag{176}$$

We can define $P_{N-1} = (Q_{N-1} + K_{N-1}^T R_{N-1}K + (A_{N-1} - B_{N-1}K_{N-1})^T Q_N(A_{N-1} - B_{N-1}K_{N-1}))$, such that:

$$V_{N-1}(x) = \frac{1}{2}x^T P_{N-1}x \tag{177}$$

- We now have a recursion in K and P that we can iterate until $k = 1$. This is just the Riccati equation again.

28.3.1 Backward Dynamic Programming Algorithm

Algorithm 6 Backward DP Algorithm

```
1: $V_N(x) \leftarrow l_N(x)$
2: $K \leftarrow N$
3: while $K > 1$ do
4:     $V_{k-1} = \min_u \left[l(x,u) + V_k(f(x,u))\right]$          ▷ The Bellman Equation.
5:     $k \leftarrow k - 1$
```

- If we know $V_k(x)$, the optimal feedback policy is:

$$u_k(x) = \arg\min_u \left[l(x_k, u) + V_{k+1}(f(x_k, u))\right] \tag{178}$$

- DP equations can be written equivalently in terms of action-value or Q functions:

$$S_k(x,u) = l(x,u) + V_{k+1}(f(x,u)) \tag{179}$$
$$\implies u_k(x_k) = \arg\min_u S_k(x_k, u) \tag{180}$$

- These are usually denoted $Q(x,u)$, be we will use S, since Q is used in the LQR state cost as well.

28.3.2 The Curse

- DP is sufficient for a global optimum.
- It is only tractable for simple problems, such as LQR problems or low-dimensional problems.
- $V(x)$ stays quadratic for LQR problems, but becomes impossible to write down analytically for even simple non-linear problems.
- Even if we could, the $\min_u S(x,u)$ will be non-convex, and possibly hard to solve on its own.
- The cost of DP blows up with state dimension, due to the difficulty of representing $V(x)$.

28.3.3 Why do we care?

- Approximate DP, where $V(x)$, or $S(x,u)$ are reprsented with function approximators can work well.
- Forms the basis of a lot of modern Reinforcement Learning.
- DP generalizes to stochastic problems well, (just wrap everything in expectation operators), whereas Pontryagin's does not.

28.3.4 What are those Lagrange Multipliers?

- Recall Riccati derivation from QPs:

$$
\begin{aligned}
\lambda_n &= P_n x_n && (181)\\
P_n &= Q + A^T P_{n+1}(A - BK) && (182)\\
&= Q + K^T R K + (A - BK)^T P_{n+1}(A - BK) && (183)\\
V_n(x) &= \frac{1}{2} x^T P_n x && (184)\\
\implies \lambda_n &= \nabla_x V_n(x) && (185)
\end{aligned}
$$

- The dynamics multipliers are the cost-to-go gradients!
- Carries over to the general non-linear setting, beyond just LQR.

28.3.5 Example

Remember to check out the lecture video for an example, where λ_n from the QP matches $\nabla_x V_n(x)$ from DP.

28.4 Convex Model-Predictive Control

- LQR is very powerful, but often need to explicitly reason about constraints.
- Often these are simple (ex. torque limits), and can be encoded as a convex set.
- Constraints break the Riccati solution, but we can still solve the QP online.
- Convex MPC has gotten extremely popular as computers have gotten faster.

28.4.1 Background: Convexity

- Convex Set: A line connecting any two points in the set is contained within the set.
- Standard examples:
  - Linear subspaces ($Ax = b$).
  - Half-spaces, boxes, and polytopes ($Ax \leq b$).
  - Ellipsoids ($x^T P x \leq 1$).
  - Cones ($||x_{2:n}||_2 \leq x_1$). This is called the second order cone, it's the usual ice-cream cone you're used to.
- Convex functions: A function $f(x) : \mathbb{R}^n \longrightarrow \mathbb{R}$ who's epigraph is a convex set.
- Examples:
  - Linear $f(x) = c^T x$.
  - Quadratic $f(x) = \frac{1}{2} x^T Q x + q^T x, \ni Q \geq 0$.
  - Norms $f(x) = |x|$.
- Convex Optimization Problem: Minimize a convex function over a convex set.
- Examples:
  - Linear Program (LP): Linear $f(x)$, linear $c(x)$.
  - Quadratic Program (QP): Quadratic $f(x)$, linear $c(x)$.
  - Quadratically Constrained (QCQP): Quadratic $f(x)$, ellipsoid $c(x)$.
  - Second-order cone program (SOCP): Linear $f(x)$, cone $c(x)$.
- Convex problems don't have spurious local optima that satisfy the KKT. If you find a local KKT solution, you have the global optimum!
- Practically, Newton's method converges really fast and reliably (5-10 iterations at maximum).
- Can bound the solution time for real-time control.