ROBOT DYNAMIC MODELING

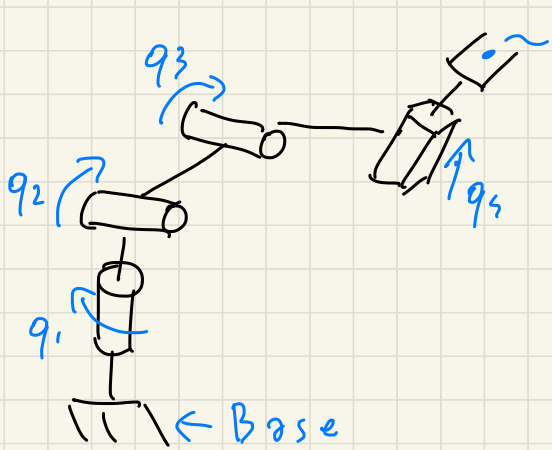
Definitions:

- Kinematics (forward)
- Statics
- Dynamics

- Homogeneous matrix
- Jacobian
- ...
- $M\ddot{q} + C\dot{q} + g(q) = \tau + J^T w$

ROBOT MANIPULATOR

- Set of rigid bodies connected by joints
  - links
  - revolute
  - prismatic
- Typically joints are actuated by system of motor + trasmission (e.g., gear box)

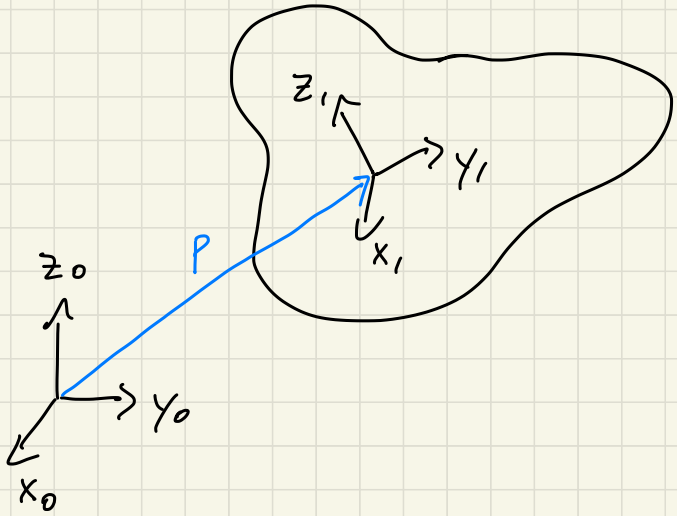


q_i: i-th joint coordinate

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix}$$

HOMOGENEOUS TRANSFORMATION

Def: pose $\triangleq$ position and orientation

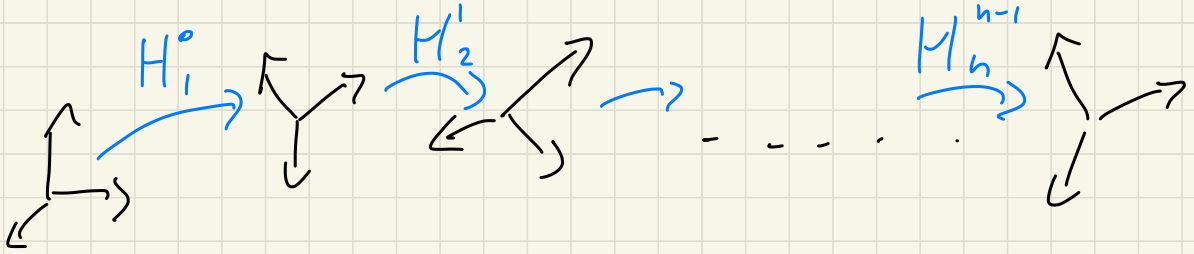


$$H_1^0 = \begin{bmatrix} R_1^0 & P \\ 0_{1\times3} & 1 \end{bmatrix}$$

$$R_1^0 = \begin{bmatrix} x_1^0 & y_1^0 & z_1^0 \end{bmatrix}$$

$$R_1^0 = R_0^{1\,-1} = R_1^{0\,T}$$

COMPOSITION OF TRANSFORMATION



$$H_n^0 = H_1^0 \, H_2^1 \, H_3^2 \ldots H_n^{n-1}$$

$H_i^{i-1}(q_i) \Rightarrow$ depends on i-th joint angle only

$$H_n^0(q_1, q_2 \ldots q_n) = H_n^0(q)$$

REPRESENTATION OF ORIENTATION

- Rotation matrix [3×3]
- Euler angles [3]
- Roll-Pitch-Yaw [3]
- Angle-axis [4]
- Quaternions [4]

} Minimal representations
Have singularities

DIFFERENTIAL KINEMATICS

GEOMETRICAL APPROACH

We want to get ${}^0V_e = \begin{bmatrix} \dot{p}_e \\ \omega_e \end{bmatrix}$ ← linear vel ← angular vel

$${}^0\dot{p}_n = {}^{n-1}\dot{p}_n + {}^0\dot{p}_{n-1} = {}^{n-1}\dot{p}_n + {}^{n-2}\dot{p}_{n-1} + {}^0\dot{p}_{n-2} = \sum_{i=0}^{n-1} {}^i\dot{p}_{i+1}$$

Rotation axis of joint i

$${}^{i-1}\dot{p}_i = \begin{cases} \text{Revolute: } \left[\hat{z}_i \times (p_i - p_{i-1})\right]\dot{q}_i = J_{p_i}\dot{q}_i \\ \text{Prismatic: } \hat{z}_i\dot{q}_i = J_{o_i}\dot{q}_i \end{cases}$$

$$J_p \triangleq \begin{bmatrix} J_{p_1} & J_{p_2} & \dots & J_{p_n} \end{bmatrix}$$

$$J_o \triangleq \begin{bmatrix} J_{o_1} & J_{o_2} & \dots & J_{o_n} \end{bmatrix}$$

$${}^0\dot{p}_n = \sum_{i=0}^{n-1} J_{p_{i+1}}\dot{q}_{i+1} = J_p\dot{q}$$

$${}^0\omega_n = \sum_{i=0}^{n-1} J_{o_i}\dot{q}_i = J_o\dot{q}$$

$${}^0V_e = \begin{bmatrix} J_p \\ J_o \end{bmatrix}\dot{q} = J(q)\dot{q}$$

Dimension: 6xN, where N is the number of joints

Jacobian matrix

DIFFERENTIAL KINEMATICS

ANALITACAL APPROACHI

$$\phi_n = \begin{bmatrix} x_n \\ y_n \\ z_n \\ \varphi_n \\ \theta_n \\ \psi_n \end{bmatrix} = \phi_n(q) \qquad \dot{\phi} = \frac{\partial \phi}{\partial q} \cdot \frac{dq}{dt} = \frac{\partial \phi}{\partial q} \dot{q}$$

Analitical Jacobian J_A

$$= \qquad \neq$$

$$\begin{bmatrix} J_P \\ J_{A_O} \end{bmatrix} = J_A \neq J = \begin{bmatrix} J_P \\ J_O \end{bmatrix}$$

STATICS

Problem: given wrench (force + moment) applied at the end-effector => compute joint torques (τ) generated by this wrench

$$w = \begin{bmatrix} f \\ m \end{bmatrix}$$

Power at the joints: $P_\tau = \tau^T \dot{q}$

Power at end-effector: $P_e = f^T \dot{p} + m^T \omega = \begin{bmatrix} f^T & m^T \end{bmatrix} \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} =$

$= w^T v = w^T J \dot{q}$

We can simplify $\dot{q}$ because we know that this is valid for each $\dot{q}$

$$\tau^T \dot{q} = w^T J \dot{q} \Rightarrow \tau^T = w^T J \Rightarrow \tau = J^T w$$

Should hold $\forall \dot{q}$

DYNAMICS

Problems:

- Direct dynamics: $q, \dot{q}, \tau \Rightarrow \ddot{q}$ SIMULATION
- Inverse dynamics: $q, \dot{q}, \ddot{q} \Rightarrow \tau$ CONTROL

$$M(q)\ddot{q} + \underbrace{C(q,\dot{q})\dot{q} + g(q)}_{h(q,\dot{q})} = \tau + J^T(q)\,w$$

- $M(q) \in \mathbb{R}^{n \times n}$: mass matrix, positive-definite, symmetric
- $C(q,\dot{q}) \in \mathbb{R}^{n \times n}$: centrifugal + Coriolis effects
- $g(q) \in \mathbb{R}^n$: gravity torques

- Nonlinear equations
- Linear in $\ddot{q}, \tau, w$ if $(q,\dot{q})$ is fixed

JOINT MOTION CONTROL

Given a reference joint trajectory $q^{ref}(t)$, compute $\tau(t)$ such that $q(t) \simeq q^{ref}(t)$.

Assumption: We know system dynamics $M(q)\ddot{q} + h(q, \dot{q}) = \tau$

IDEA #1: OPEN-LOOP CONTROL

- Compute $\dot{q}^{ref}(t)$, $\ddot{q}^{ref}(t)$
- Compute $\tau(t) = M(q^{ref}(t))\,\ddot{q}^{ref}(t) + h(q^{ref}(t), \dot{q}^{ref}(t))$
- **Assuming** $q(0) = q^{ref}(0)$, $\dot{q}(0) = \dot{q}^{ref}(0)$ then

$$M(q)\ddot{q} + h(q,\dot{q}) = \tau = M(q^{ref})\ddot{q}^{ref} + h(q^{ref}, \dot{q}^{ref}) \Rightarrow \ddot{q} = \ddot{q}^{ref}$$

#2: PID CONTROL

$$\tau(t) = K_p\left(q^{ref}(t) - q(t)\right) + K_d\left(\dot{q}^{ref}(t) - \dot{q}(t)\right) + K_i \int_0^t \left(q^{ref}(s) - q(s)\right) ds$$

K_p, K_d, K_i are diagonal positive-definite matrices

Guarantee that $q(t) \Rightarrow q^{ref}(t)$ if $\dot{q}^{ref} = 0$

$$M\ddot{q} + C\dot{q} + g = \tau = K_p e + K_d \dot{e} + K_i \int e\, ds$$

At steady state ($\dot{q} = \ddot{q} = 0$) I have

$$g = K_p e + K_i \int e\, ds \quad \Rightarrow \quad K_p \dot{e} + K_i e = 0$$

Stable first-order linear dynamical system

$$\boxed{\dot{e} = -K_i^{-1} K_p e}$$

$e \to 0$

$q \to q^{ref}$

If $K_I = 0 \Rightarrow K_P e = g \Rightarrow e = K_P^{-1} g$

#3: PD + GRAVITY COMPENSATION

$$\tau(t) = K_p \underbrace{(q^{ref}(t) - q(t))}_{e} + K_d \underbrace{(\dot{q}^{ref}(t) - \dot{q}(t))}_{\dot{e}} + g(q(t))$$

Better performance than PID alone.

Steady-state analysis:

$$g = K_p e + g \quad \Rightarrow \quad K_p e = 0 \quad \Rightarrow \quad e = 0$$

#4: INVERSE DYNAMICS CONTROL

A.K.A. FEEDBACK LINEARIZATION

COMPUTED TORQUE

$$\tau = M(\ddot{q}^{ref} + K_p e + K_d \dot{e}) + h$$

System closed-loop dynamics:

$$M\ddot{q} + h = M(\ddot{q}^{ref} + K_p e + K_d \dot{e}) + h$$

$$\ddot{q} = \ddot{q}^{ref} + K_p e + K_p \dot{e}$$

$$\ddot{e} + K_d \dot{e} + K_p e = 0$$

$$y \triangleq \begin{bmatrix} e \\ \dot{e} \end{bmatrix} \qquad \dot{y} = \begin{bmatrix} \dot{e} \\ \ddot{e} \end{bmatrix}$$

$$\dot{y} = \begin{bmatrix} 0 & I \\ -K_p & -K_d \end{bmatrix} y$$

#VARIANT OF INVERSE DYNAMICS

$$\tau = M\ddot{q}^{ref} + h + K_p e + K_d \dot{e}$$

Closed-loop behavior:

$$M\ddot{q} + \cancel{h} = M\ddot{q}^{ref} + \cancel{h} + K_p e + K_d \dot{e}$$

$$\ddot{q} = \ddot{q}^{ref} + \tilde{M}^{-1} K_p e + \tilde{M}^{-1} K_d \dot{e}$$

$$\ddot{q}^{ref} - \ddot{q} + \bar{K}_p e + \bar{K}_d \dot{e} = 0$$

$$\bar{K}_p \triangleq \tilde{M}^{-1} K_p > 0$$

$$\bar{K}_d \triangleq \tilde{M}^{-1} K_d > 0$$

Positive definite

Holds because Kp and Kd are diagonal

CARTESIAN-SPACE MOTION CONTROL

Given a reference trajectory for end-effector $x^{ref}(t)$, compute $\tau(t)$ such that $x(t) \simeq x^{ref}(t)$.

[Assume $x \in \mathbb{R}^3$ represents the e-e position]

IDEA #1: Compute $q^{ref}(t)$ such that $x^{ref}(t) = FG(q^{ref}(t))$

Inverse Kinematics

You cannot do this computation offline

where $FG(\cdot)$ is forward geometry function.

Then apply joint motion control.

OPERATIONAL SPACE CONTROL

Method to do cartesian space control

Operational space dynamics

$$J\bar{M}^{-1}\left[\underline{M\ddot{q} + h = \tau}\right]$$

Dynamics of the robot in joint space

$$J\ddot{q} + J\bar{M}^{-1}h = J\bar{M}^{-1}\tau$$

$$\ddot{x} - \dot{J}\dot{q} + J\bar{M}^{-1}h = J\bar{M}^{-1}\tau$$

$$\Lambda\,\ddot{x} + \Lambda(J\bar{M}^{-1}h - \dot{J}\dot{q}) = \Lambda J\bar{M}^{-1}\tau$$

$$\Lambda\,\ddot{x} + \mu = \Lambda J\bar{M}^{-1}\tau$$

$$\Lambda(q)\,\ddot{x} + \mu(q,\dot{q}) = f$$

(Approfondimento su Jacobian)

J = end-effector Jacobian

$$J = \frac{dFG(q)}{dq}$$

$$x = FG(q)$$

$$\dot{x} = J\dot{q}$$

$$\ddot{x} = J\ddot{q} + \dot{J}\dot{q}$$

Pseudoinverse

$$\bar{\Lambda} = (J\bar{M}^{-1}J^T + \varepsilon I)^{-1} \qquad \varepsilon > 0 \quad \varepsilon = 10^{-5}$$

$$\Lambda \triangleq (J\bar{M}^{-1}J^T)^{-1}$$

Assume $\tau = J^T f$

$$\left[\Lambda J\bar{M}^{-1}J^T = I\right]$$

ASSUME J is full row rank

Chose $f = \Lambda \underbrace{\left(\ddot{x}^{ref} + K_p(x^{ref} - x) + K_d(\dot{x}^{ref} - \dot{x}) \right)}_{\ddot{x}^d} + \mu$

$\tau = J^T f = J^T(\Lambda \ddot{x}^d + \mu) =$

$= J^T \Lambda \ddot{x}^d + J^T \Lambda (J \tilde{M}^{-1} h - \dot{J} \dot{q})$

If I apply this control law I get $\ddot{x} = \ddot{x}^d$

A simplified version of this control law is:

$\tau = J^T \Lambda \ddot{x}^d + h$

Let's check it works:

$M \ddot{q} + h = J^T \Lambda \ddot{x}^d + h$

$$J\ddot{q} + \cancel{J\bar{M}^{-1}h} = \cancel{J\bar{M}^{-1}J^{T}\Lambda}\,\ddot{x}^{d} + \cancel{J\bar{M}^{-1}h}$$

$$J\ddot{q} = \ddot{x}^{d}$$

$$\ddot{x} - \dot{J}\dot{q} = \ddot{x}^{d} \quad \Rightarrow \quad \ddot{x} \simeq \ddot{x}^{d}$$

$\dot{J}\dot{q}$: Typically small

REDUNDANCY

$$\Lambda \ddot{x} + \mu = \left(J \tilde{M}^{-1} J^T\right)^{-1} J \tilde{M}^{-1} \tau = \bar{J}^{T\#} \tau$$

$\bar{J}^{T\#}$ is the pseudo-inverse of J^T because $\bar{J}^{T\#} J^T = I$

$J^T \bar{J}^{T\#} \neq I$

Any τ_0 such that $\bar{J}^{T\#} \tau_0 = 0$ does not affect $\ddot{x}$.

$$\Lambda \ddot{x} + \mu = \bar{J}^{T\#} (\tau + \tau_0) = \bar{J}^{T\#} \tau$$

I can compute τ_0 as $\tau_0 = (I - J^T \bar{J}^{T\#}) \tau_1$

Verify $\bar{J}^{T\#} \tau_0 = \bar{J}^{T\#} (I - J^T \bar{J}^{T\#}) \tau_1 = (\bar{J}^{T\#} - \bar{J}^{T\#} J^T \bar{J}^{T\#}) \tau_1$

$= 0$

So I can extend my control law as:

$$\tau = \bar{J}^T \Lambda \ddot{x}^d + h + (I - \bar{J}^T \bar{J}^{T\#}) \tau_1$$

How do I compute τ_1?

If I have a refence posture q^{ref} I can compute τ_1 as

$$\tau_1 = M\left(K_p(q^{ref} - q) - K_d \dot{q}\right)$$

INVERSE KINEMATICS + INVERSE DYNAMICS

$$\ddot{x} = J\ddot{q} + \dot{J}\dot{q} \quad \Rightarrow \quad \ddot{q}^d = J^{\#}(\ddot{x}^d - \dot{J}\dot{q})$$

$$\left[J J^{\#} = I \right]$$

$$\tau = M\ddot{q}^d + h$$

$$J^{\#} = J^T (J J^T)^{-1}$$

Verify closed-loop behavior:

$$M\ddot{q} + h = M\ddot{q}^d + h$$

$$\ddot{q} = J^{\#}(\ddot{x}^d - \dot{J}\dot{q})$$

$$J\ddot{q} = J J^{\#}(\ddot{x}^d - \dot{J}\dot{q})$$

$$J\ddot{q} + \dot{J}\dot{q} = \ddot{x}^d \quad \Rightarrow \quad \ddot{x} = \ddot{x}^d$$

ASSUMPTION:

J is full row rank

All rows are linearly independent

Q.P.-BASED REACTIVE CONTROL

- Joint space control
- Review of convex optimization
- Task-Space control
- Underactuated robots
- Contact constraints
- Multi-Task control

JOINT-SPACE INVERSE DYNAMICS

$$M(q)\ddot{q} + h(q,\dot{q}) = \tau$$

Given $q^r(t)$, Find $\tau(t)$ such that $q(t) \simeq q^r(t)$

$$\tau^d = M\ddot{q}^d + h \qquad \ddot{q}^d = \ddot{q}^r + K_d(\dot{q}^r - \dot{q}) + K_p(q^r - q)$$

$$(\tau^*, \ddot{q}^*) = \underset{\tau,\ddot{q}}{\arg\min} \ \frac{1}{2}\|\ddot{q} - \ddot{q}^d\|^2 = \frac{1}{2}\ddot{q}^T\ddot{q} - \ddot{q}^{dT}\ddot{q} + \cancel{\frac{1}{2}\ddot{q}^{dT}\ddot{q}^d}$$

constant

subject to $M\ddot{q} + h = \tau$

$$\left.\begin{aligned} \tau^* &= \tau^d = M\ddot{q}^d + h \\ \ddot{q}^* &= \ddot{q}^d \end{aligned}\right\} \text{SOLUTION}$$

TORQUE, CURRENT, VELOCITY LIMITS

$$\underset{\tau,\ddot{q}}{\text{minimize}} \quad \frac{1}{2}\|\ddot{q} - \ddot{q}^d\|^2$$

subject to $M\ddot{q} + h = \tau$

$$\tau^{min} \leq \tau \leq \tau^{max}$$

$$i^{min} \leq K_\tau \tau \leq i^{max}$$

$$\dot{q}^{min} \leq \dot{q} + \Delta t\, \ddot{q} \leq \dot{q}^{max}$$

We know current is proportional to torque:

$$i = K_\tau \tau$$

Assuming $\ddot{q}$ remains constant during time step Δt:

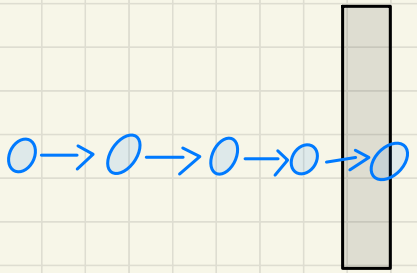
$$\dot{q}(t + \Delta t) = \dot{q}(t) + \Delta t\, \ddot{q}$$

JOINT POSITION LIMITS

Could use same approach as for joint vel.:

$$q(t+\Delta t) = q(t) + \Delta t\, \dot{q}(t) + \frac{\Delta t^2}{2} \ddot{q}$$

$$q^{min} \leq q + \Delta t\, \dot{q} + \frac{\Delta t^2}{2} \ddot{q} \leq q^{max}$$



This could result in large accelerations, incompatible with torque/current limits => QP unfeasible

Heuristic: use larger values of Δt (e.g. [10, 100] ms) improve behavior, but still no guarantees.

TAXONOMY OF CONVEX OPTIMIZATION PROBLEMS

- **Least-Squares Program (LSP)**
  - Linear eq./ineq. constraints $(Ax \leq b,\ Ax = b)$
  - 2-norm of affine function $\|Ax - b\|^2$

- **LSP are subclass of Quadratic Programs (QP):**
  - Linear eq./ineq. constraints
  - Convex quadratic cost: $\frac{1}{2} x^T H x + g^T x$, $H \succeq 0$ (Positive Semi Definite)

$$\text{eig}(H) \geq 0$$

LSP and QP are convex optim. problems and can be solved efficiently with off-the-shelf software.

(< 1 ms)

LEAST-SQUARES PROGRAMS

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|Ax-b\|^2 = \frac{1}{2}x^T A^T A x - b^T A x + \frac{1}{2}b^T b \triangleq f(x)$$

$H \succeq 0$

$\nabla f = A^T A x - A^T b = 0 \Rightarrow$ if $A^T A$ is invertible

$$x^\star = (A^T A)^{-1} A^T b$$

$(A^T A)^{-1} A^T A = I$ (left) pseudo-inverse of A

If $A^T A$ not invertible, but AA^T is invertible:

$$x^\star = A^T (AA^T)^{-1} b \Leftarrow \begin{cases} Ax^\star = AA^T(AA^T)^{-1}b = b \\ A^T \cancel{A A^T}\cancel{(AA^T)^{-1}} b - A^T b = A^T b - A^T b = 0 \end{cases} \Rightarrow \begin{cases} f(x) = 0 \\ \nabla f = 0 \end{cases}$$

(right) pseudo-inverse

(MOORE-PENROSE) PSEUDO-INVERSE

$$x^* = \arg\min_x \|Ax - b\|^2 \iff x^* = A^+ b$$

If A is full row rank $\Rightarrow$ $A^+ = A^T(AA^T)^{-1}$

If A is full column rank $\Rightarrow$ $A^+ = (A^TA)^{-1}A^T$

If A is not full rank $\Rightarrow$ A^+ still exists, but equations above can't be used. In practice A^+ is often computed using numerical methods, such as Singular Value Decomposition (SVD).

FROM JOINT TO CARTESIAN SPACE

Given a reference trajectory $x^r(t)$ for the e-e, find $\tau(t)$ such that $x(t) \simeq x^r(t)$. We know that:

$$\ddot{x} = J\ddot{q} + \dot{J}\dot{q} \qquad \ddot{x}^d = \ddot{x}^r + K_d(\dot{x}^r - \dot{x}) + K_p(x^r - x)$$

We can compute desired joint acc. $\ddot{q}^d$ as:

$$\begin{cases} \ddot{q}^d = J^+(\ddot{x}^d - \dot{J}\dot{q}) \\ \tau^d = M\ddot{q}^d + h \end{cases} \qquad J^+ = J^T(JJ^T)^{-1}$$

↑ Assuming J is full-row rank

$$(\tau^*, \ddot{q}^*) = \underset{\ddot{q}, \tau}{\arg\min} \; \| \overbrace{J\ddot{q} + \dot{J}\dot{q}}^{\ddot{x}} - \ddot{x}^d \|^2 \equiv \| \ddot{x} - \ddot{x}^d \|^2$$

$$\text{subject to} \quad M\ddot{q} + h = \tau$$

TASK MODELS

- Generalize concept of end-effector control
- Task = control objective
- Describe task with function $e(\cdot)$ to minimize
- Assume that $e(\cdot)$ describes error btw real & reference traj.:

$$e(x, u, t) = \underbrace{y(x, u)}_{\text{real}} - \underbrace{y^r(t)}_{\text{reference}}$$

$$x \triangleq (q, \dot{q})$$
$$u \triangleq \tau$$

- We cannot directly minimize $e(\cdot)$ if it depends on q and $\dot{q}$, which are not decision variables of the LSP.

IDEA: Impose dynamics of e (e.g., $\dot{e}$ or $\ddot{e}$) such that:

i) $\lim_{t \to \infty} e(t) = 0$

ii) Dynamics of e is affine function of $(\ddot{q}, \tau)$

CASE #1: VELOCITY TASK FUNCTION

$$e(x, u, t) = e(\dot{q}, t) = y(\dot{q}) - y^r(t)$$

For instance: [$y(\dot{q})$ = angular momentum]

$$\dot{e} = \dot{y} - \dot{y}^r = \frac{\partial y}{\partial \dot{q}} \frac{d\dot{q}}{dt} - \dot{y}^r = J\ddot{q} - \dot{y}^r$$

Choose $\dot{e}$ such that $e \to 0$. Let's take a linear feedback:

$$\dot{e} = -Ke \qquad K > 0 \quad \Rightarrow \quad \lim_{t \to \infty} e(t) = 0$$

$$J\ddot{q} - \dot{y}^r = -Ke$$

$$\underbrace{J}_{A}\ddot{q} = \underbrace{\dot{y}^r - Ke}_{b} \quad \Rightarrow \quad A\ddot{q} - b = 0$$

CASE #2: CONFIGURATION TASK FUNCTION

$$e(x, u, t) = e(q, t) = y(q) - y^r(t)$$

$$\dot{e} = \dot{y} - \dot{y}^r = \frac{\partial y}{\partial q}\frac{dq}{dt} - \dot{y}^r = J\dot{q} - \dot{y}^r$$

$$\ddot{e} = J\ddot{q} + \dot{J}q - \ddot{y}^r$$

Let's impose linear dynamics: $\ddot{e} = -K_p e - K_d \dot{e}$ $\quad K_p \succ 0$, $K_d \succ 0$

$$J\ddot{q} + \dot{J}\dot{q} - \ddot{y}^r = -K_p e - K_d \dot{e}$$

$$J\ddot{q} = \ddot{y}^r - \dot{J}\dot{q} - K_p e - K_d \dot{e}$$

$$A\ddot{q} = b \qquad \Rightarrow \text{minimize } \|A\ddot{q} - b\|^2$$

CASE #3: CONTROL-INPUT TASK FUNCTION

Control inputs $u \triangleq \tau$ are part of the decision variables.

Therefore $e(x, u, t) = e(u, t)$ must be affine:

$$e(u, t) = A_u(t)\, u - b(t)$$

If it's not, I cannot use it in a Q.P.-based controller.

TASK FUNCTIONS: SUMMARY

3 kinds of task functions:

- Affine function of $u \triangleq \tau$
- Nonlinear function of x:
  - For functions of $\dot{q} \Rightarrow$ impose first derivative $\dot{e}$
  - For functions of $q \Rightarrow$ impose 2nd derivative $\ddot{e}$

In the end we have affine functions of $(\ddot{q}, \tau)$:

$$g(z) = \underbrace{\begin{bmatrix} A_x & A_u \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \ddot{q} \\ \tau \end{bmatrix}}_{z} - b$$

TASK-SPACE INVERSE DYNAMICS

$$\underset{z=(\ddot{q},\tau)}{\text{minimize}} \quad \frac{1}{2}\|Az-b\|^2$$

$$\text{subject to} \quad [M \;\vdots\; -I]\, z = -h$$

- Joint-space control: $A=[I \;\; 0]$, $b=\ddot{q}^d$

$$Az-b=\ddot{q}-\ddot{q}^d$$

- End-effector control: $A=[J \;\; 0]$, $b=\ddot{x}^d-\dot{J}\dot{q}$

$$Az-b=J\ddot{q}+\dot{J}\dot{q}-\ddot{x}^d$$

UNDER-ACTUATED SYSTEMS

Example: remove one motor from manipulator. How does it affect dynamics?

$$M\ddot{q} + h = \tau = \begin{bmatrix} \tau_0 \\ \vdots \\ \tau_{n-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \tau_1 \\ \vdots \\ \tau_{n-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \tau_{1:n} \end{bmatrix}$$

Introduce $S \triangleq \begin{bmatrix} 0 & I \end{bmatrix}$ $\quad M\ddot{q} + h = S^T \tau_{1:n}$

Examples of underactuated systems:

- legged robots (biped, quadruped)
- wheeled robots
- flying robots (most of them)
- underwater robots (most of them)

TSID FOR UNDERACTUATED ROBOTS

$$\underset{z=(\ddot{q},\tau)}{\text{minimize}} \quad \frac{1}{2}\|Az-b\|^2$$

$$\text{subject to} \quad [M \quad \cancel{-I}\ ]\, z = -h$$

$$-S^T$$

TSID FOR RIGID CONTACT

Rigid contacts constrain the motion:

$$J \triangleq \frac{\partial c(q)}{\partial q}$$

$c(q) = \text{const} \iff$ contact points don't move

$J\dot{q} = 0 \iff$ contact point velocities are null

$\dot{J}\dot{q} + J\ddot{q} = 0 \iff$ contact point accelerations are null

Introduce contact forces f as decision variables in QP:

$$\text{minimize}_{z=(\ddot{q},\tau,f)} \quad \|Az - b\|^2$$

$$\text{subject to} \begin{cases} M\ddot{q} + h = S^T\tau + J^T f \\ J\ddot{q} + \dot{J}\dot{q} = 0 \end{cases} \iff \begin{bmatrix} M & -J^T & -S^T \\ J & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -h \\ -\dot{J}\dot{q} \end{bmatrix}$$

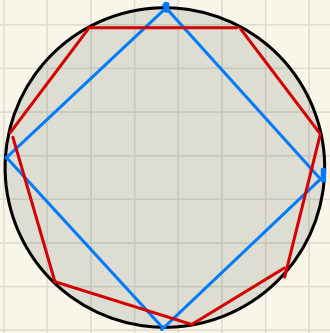
FRICTION CONE CONSTRAINTS

$$\| f_T \|^2 \leq \mu f_N^2$$

f_T: Tangential Force

f_N: Normal Force

μ = friction coefficient



Approximate quadratic cone with linear pyramidal cone:

$$Bf \leq 0$$

Include this in TSID's QP.

MULTI-TASK CONTROL

For complex redundant robots typically one has multiple control objectives, e.g. N tasks, each defined by a task function

$$g_i(z) \triangleq \| A_i z - b_i \|^2 \qquad i = 1 \dots N$$

TSID QP:

$$\begin{aligned} &\text{minimize}_{z=(\ddot{q},\tau,f)} && \sum_{i=1}^{N} w_i \, g_i(z) = \| A z - b \|^2 \\ &\text{subject to} && \begin{bmatrix} M & -J^T & -S^T \\ J & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -h \\ -\dot{J}\dot{q} \end{bmatrix} \end{aligned}$$

$w_i \in \mathbb{R}^+ \triangleq$ user-defined weight

Large $w_i \Rightarrow$ high importance to task i

OPTIMAL CONTROL

XVII century, Bernoulli (1696): ground shape to let ball go from A to B in minimum time? Brachistochrone curve

- How to accelerate car to reach goal in min time/energy with max speed

CLASSIC CONTROL:

- need reference trajectory
- rise time (time to reach 90% of target)
- overshoot
- settling time (time to reach an equilibrium)
- hard to handle constraints

O.C.:

- no need for ref. traj.
- objective clearly specified through cost function
- can handle constraints

Running cost

Fixed length of the problem

O C P

cost function

minimize $\int_0^T \ell(x(t), u(t), t)\,dt + \ell_f(x(T))$

Terminal cost depends only on the terminal state

$x(\cdot)$, $u(\cdot)$ — decision variables

subject to

Differential equation

Dynamics (equation of motion)

$\dot{x}(t) = f(x(t), u(t), t)$ $\quad \forall t \in [0, T]$

$x(0) = x_0$ — Initial conditions (It's not mandatory)

Physical limits

$g(x(t), u(t), t) \leq 0$ $\quad \forall t \in [0, T]$

Path constraints (e.g., actuator limits)

optimal control problem

- $x(\cdot)$, $u(\cdot)$ are trajectories => infinite-dimensional objects
- constraints are infinitely many! (you have infinite time instant between $[0, T]$)
- $\neq$ optimization problem

Main differences with optimization problem

OCP EXAMPLE: PENDULUM

terminal cost must be in function only of the terminal state

$$\text{minimize}_{x(\cdot),\,u(\cdot)} \int_0^T \left(w_q q(t)^2 + w_{\dot q} \dot q(t)^2 + w_\tau \tau(t)^2 \right) dt + 10^2\, q(T)^2$$

pendulum reach $q(T)=0$ faster → terminal condition

torque and acc. are linear dependent so we can optimize only one

standard form

It's possible to use Absolute value instead of square → with square you are more tolerant with small errors and penalize more big errors

subject to

$$I\ddot q = \tau + mg\sin(q) \Rightarrow \begin{bmatrix} \dot q \\ \ddot q \end{bmatrix} = \begin{bmatrix} \dot q \\ I^{-1}(\tau + mg\sin(q)) \end{bmatrix} \quad \forall t$$

I: inertia; $mg\sin(q)$: gravity effect; non-linear term

system dynamic of a pendulum

$\dot x$ = $f(x, u)$

$$|\tau(t)| \le \tau^{max} \qquad \forall t \in [0, T]$$

$$q^{min} \le q(t) \le q^{max} \qquad \forall t \in [0, T]$$

$q(T) = \emptyset$ (model choice)

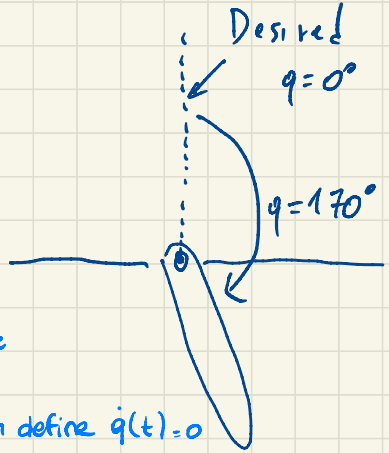
If I don't specify a terminal constraint, I need to add a weight to the terminal cost. We choose to use one or the other.

↳ Same for joint velocities, we can define $\dot q(t) = 0$

to set the final position to joint angle $q=0$

$x = (q, \dot q)$ ← state of the system

$u = \tau$ ← control input



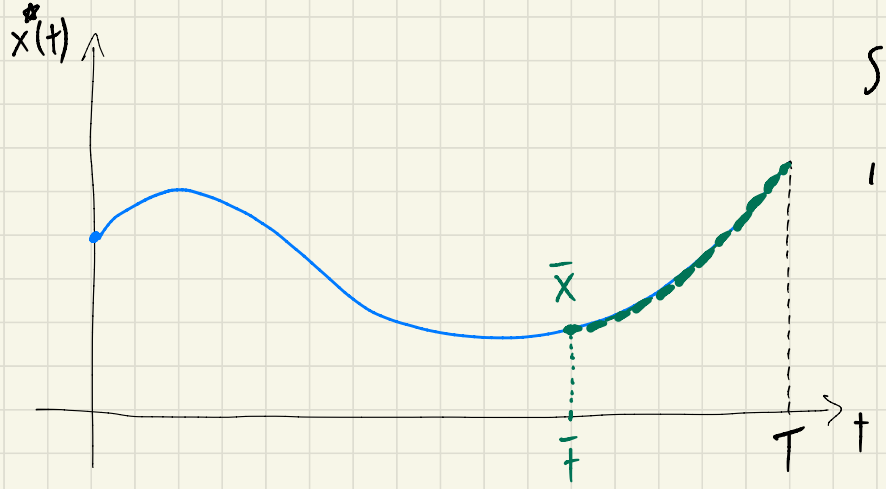
O.C. METHOD FAMILIES

	OPTIMIZE => DISCRETIZE Continuous Time	DISCRETIZE => OPTIMIZE Discrete Time
Global	(2) Hamilton-Jacobi-Bellman (HJB)	(1) Dynamic Programming (DP)
Local	(3) • Pontryagin Maximum Principle (PMP) • Calculus of variations • Indirect method	(4) • Direct method

NOTEBOOK numerical_optim

① DP: PRINCIPLE OF OPTIMALITY (BELLMAN)

Any subarc of an optimal trajectory is optimal



Subarc on $[\bar{t}, T]$ is optimal for initial value $\bar{x}$.

DP: DISCRETE TIME OCP

$$\underset{X,U}{\text{minimize}} \quad \sum_{i=0}^{N-1} l(x_i, u_i) \qquad \text{subject to} \quad x_{i+1} = f(x_i, u_i) \quad i = 0 \dots N-1$$

[No terminal cost for sake of simplicity]

$$X = (x_1, \dots, x_N)$$

$$U = (u_0, \dots, u_{N-1})$$

Take M s.t. $0 \leq M < N$. Then split problem cost:

$$\min_{X,U} \left[\sum_{i=0}^{M-1} \left(l(x_i, u_i) + \mathcal{I}(x_{i+1} - f(x_i, u_i)) \right) + \sum_{i=M}^{N-1} \left(l(x_i, u_i) + \mathcal{I}(x_{i+1} - f(x_i, u_i)) \right) \right]$$

(indicator function)

$$\underset{X,U}{\text{minimize}} \left[c_0(X_{1:M}, U_{0:M-1}) + c_M(x_M, X_{M+1:N}, U_{M:N-1}) \right]$$

$$\underset{X_{1:n},\, U_{0:M-1}}{\text{minimize}} \left[c_0(X_{1:M}, U_{0:M-1}) + \underset{X_{M+1:N},\, U_{M:N-1}}{\text{minimize}} \left[c_M(x_M, X_{M+1:N}, U_{M:N-1}) \right] \right]$$

x_M is treated as constant in this minimization!

DP: DISCRETE TIME OCP

$$\underset{X_{1:M},\,U_{0:M-1}}{\text{minimize}} \left[c_0(X_{1:M}, U_{0:M-1}) + \underbrace{\underset{X_{M+1,N},\,U_{M:N-1}}{\text{minimize}} \left[c_M(x_M, X_{M+1:N}, U_{M:N-1}) \right]}_{V_M(x_M)} \right]$$

Optimal cost of inner minimization is function of its initial state x_M

$$V_0(x_0) = \underset{X_{1:M},\,U_{0:M-1}}{\text{minimize}} \; c_0(X_{1:M}, U_{0:M-1}) + V_M(x_M)$$

VALUE FUNCTION or OPTIMAL COST-TO-GO

$V_M(x_M)$ = cost paid starting in x_M at time step M and behaving optimally

$$V_i(x_i) = \underset{u_i}{\text{minimize}} \left[\ell(x_i, u_i) + V_{i+1}\left(\overbrace{f(x_i, u_i)}^{x_{i+1}} \right) \right]$$

RECURSIVE FORMULATION

$M = 1$

DP: ALGORITHM

Given discrete-time finite-horizon OCP

$$V_N(x_N) = \ell_f(x_N)$$

Use recursive optimality principle (backward in time):

$$V_i(z) = \underset{u}{\text{minimize}} \quad \ell(z,u) + V_{i+1}(f(z,u)) \qquad i = N-1, \dots, 0$$

Once you have $V_i(\cdot)$ $\forall\, i \in [0,N]$ compute optimal control as:

$$u_i^*(x) = \underset{u}{\arg\min} \; \underbrace{\ell(x,u) + V_{i+1}(f(x,u))}_{\text{Q function}}$$

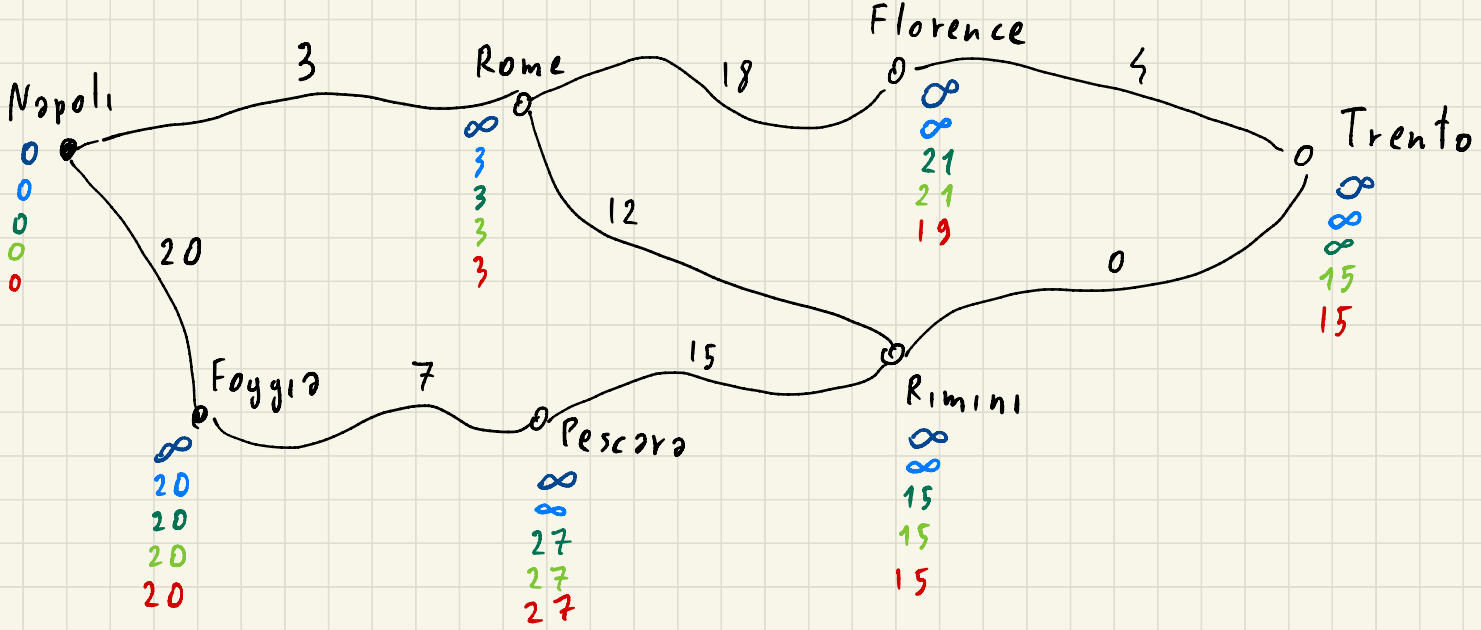
OPTIMAL FEEDBACK CONTROL POLICY

PARAMETRIC OPTIMIZ. $\neq$ NUMERICAL OPTIMIZAT.
Solution is a function not a value

DP EXAMPLE: ROAD TRIP

$V(K, city) \triangleq$ optimal cost-to-go from "city" on day K (10 days max)

$V(10,\cdot) \to V(9,\cdot) \to V(8,\cdot) \to V(7,\cdot) \to V(6,\cdot)$



DP: DISCUSSION

- Global solution for discrete-time OCP
- Feedback solution (closed-loop)

- Applicable only in specific cases
  1) discrete state-ctrl space ⇒ parametric optimization replaced by numerical optimization applied to all possible state values, results stored in lookup table
     - ⊖ Curse of dimensionality ⇒ applicable up to 2-4 states-controls
     - ⊕ Often applied to games (chess) or small continuous systems
     - ⊕ Easy to handle integer variables

  2) Linear dynamics & quadratic cost ⇒ LQR

O.C. METHOD FAMILIES

	Continuous Time	Discrete Time
Global	② Hamilton-Jacobi-Bellman (HJB)	① Dynamic Programming (DP)
Local	③ • Pontryagin Maximum Principle (PMP) • Calculus of variations • Indirect method	④ • Direct method

FROM ① DP ⇒ TO ② HJB

$$V(t_i, x) = \min_u \; l_d(x,u) + V(t_{i+1}, f_d(x,u))$$

$$l_d(x,u) = l(x,u)\,\delta$$

For small time step $\delta \to 0$ we can approximate V with Taylor expansion:

$$V(t_i, x) = \min_u \; l_d(x,u) + V(t_i, x) + \frac{\partial V}{\partial t}\delta + \frac{\partial V}{\partial x}\frac{dx}{dt}\delta$$

$$0 = \min_u \; l(x,u)\,\delta + \dot{V}\,\delta + \nabla_x V\, f(x,u)\,\delta$$

$$\dot{V} \triangleq \frac{\partial V}{\partial t}$$

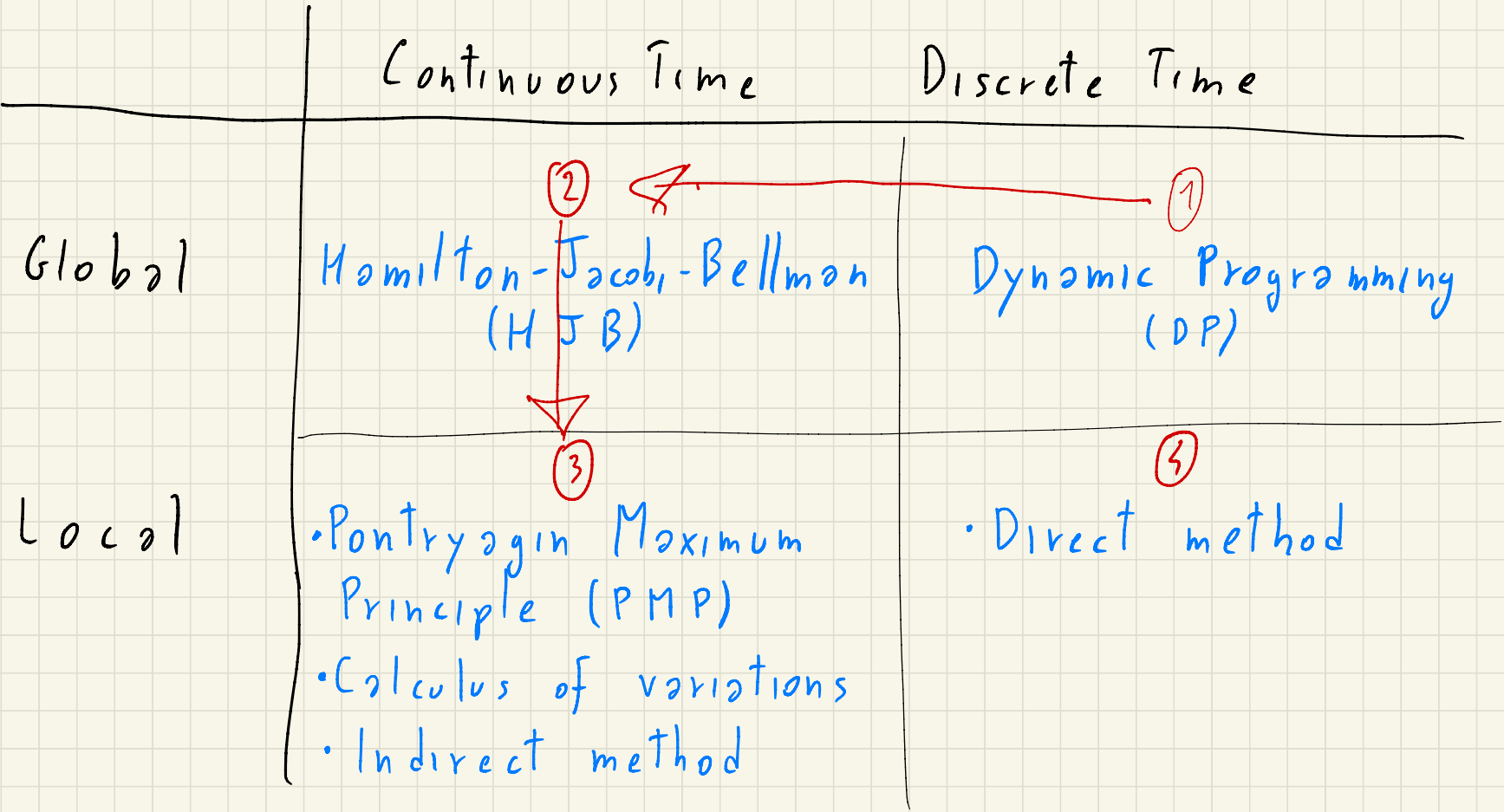
$$-\dot{V} = \min_u \; l(x,u) + \nabla_x V\, f(x,u)$$

$$V(t_f, x) = l_f(x)$$

Partial Differential Equation ⇒ Hard to solve!

Analytic solution for linear dynamics and quadratic cost

O.C. METHOD FAMILIES



PMP

HJB: $-\dot{V} = \min_u \; \ell(x,u) + \nabla_x V \, f(x,u)$

OBSERVATION: u^* depends only on $\nabla_x V$, not on V

IDEA: Introduce adjoint variables:

$$\lambda(t) \triangleq \nabla_x V(t, x(t))^T$$

Then compute u^* as:

$$u^*(t,x,\lambda) = \arg\min_u \; \ell(x,u) + \lambda^T f(x,u)$$

Hamiltonian $H(x,u,\lambda)$

How do we compute $\lambda(t)$? Differentiate HJB w.r.t. x:

$$-\nabla_x \dot{V} = \nabla_x \left[\min_u H(x,u,\nabla_x V) \right]$$

$$-\dot{\lambda} = \nabla_x H(x,u^*,\lambda) = \nabla_x \ell(x,u^*) + \lambda^T \nabla_x f(x,u^*)$$

Likewise, differentiate $V(t_f, x) = \ell_f(x)$ to get:

$$\nabla_x V(t_f, x) = \nabla_x \ell_f$$

$$\lambda(t_f) = \nabla_x \ell_f \quad \leftarrow \text{TERMINAL CONDITIONS}$$

To summarize optimality conditions as boundary value problem:

BVP

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

$$-\dot{\lambda} = \nabla_x H(x, u^*, \lambda)$$

$$u^* = \arg\min_u H(x, u, \lambda)$$

$$x(0) = x_0$$

$$\lambda(t_f) = \nabla_x \ell_f(x(t_f))$$

HOW DO WE SOLVE THIS?

SOLVING B.V.P.

- Assume we know $\lambda(0)$ (even if we know $\lambda(t_f) = \nabla_x \ell_f(x(t_f))$)
- We do know $x(0) = x_0$
- Loop:
  - Compute $u^* = \arg\min_u H(x, u, \lambda)$
  - Compute $\dot{\lambda} = -\nabla_x H(x, u^*, \lambda)$
  - Compute $\dot{x} = f(x, u^*)$
  - Integrate $\dot{x}, \dot{\lambda}$ to get new values of x, λ:

    $x := x + \Delta t\, \dot{x}$

    $\lambda := \lambda + \Delta t\, \dot{\lambda}$

But in reality we don't know $\lambda(0)$!

SOLVING B.V.P.

- Guess $\lambda(0) = \lambda_0$
- Integrate forward to get $\lambda(t_f)$
- If $\lambda(t_f) = \nabla_x \ell_f(x(t_f)) \Rightarrow$ DONE!
- Otherwise, we can see $\lambda(t_f)$ as a function of λ_0 and solve these nonlinear equations with Newton:

$$\underbrace{\lambda(t_f, \lambda_0) - \nabla_x \ell_f(x(t_f))}_{r(\lambda_0)} = 0$$

$$r(\lambda_0) + \nabla r \, \Delta\lambda_0 = 0 \Rightarrow \Delta\lambda_0 = -\nabla r^{-1} r(\lambda_0)$$

How to compute this?

Newton step

O.D.E. SENSITIVITIES

$\nabla r = \dfrac{\partial \lambda(t_f, \lambda_0)}{\partial \lambda_0}$ captures how $\lambda(t_f)$ changes for small variation of λ_0

Many techniques to compute ∇r:

- External numerical differentiation
- Variational Differential Equations
- Automatic Differentiation
- Internal Numerical Differentiation

- perturb $\lambda_0 \Rightarrow \bar{\lambda}_0 = \lambda_0 + \varepsilon e_i$
- call integrator with $\lambda(0) = \bar{\lambda}_0$
- compute derivatives by finite differences:

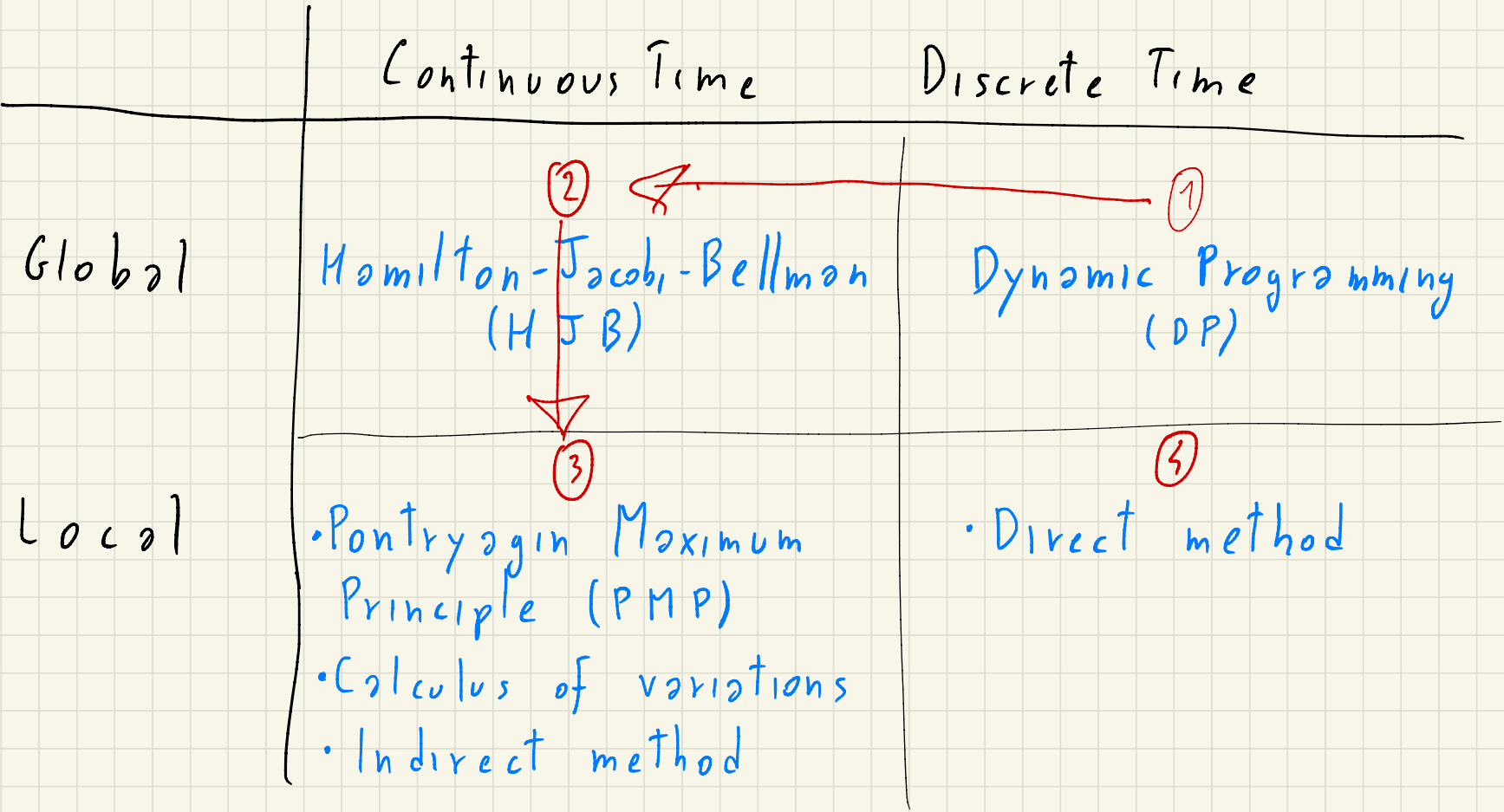
$$\frac{\lambda(t_f, \bar{\lambda}_0) - \lambda(t_f, \lambda_0)}{\varepsilon}$$

⊕ Easy to implement

⊖ Computationally expensive

⊖ Poor accuracy

O.C. METHOD FAMILIES



NONLINEAR PROGRAMMING

$$\min_x \; f(x)$$

$$\text{s.t.} \quad g(x) = 0$$

$$h(x) \leq 0$$

Feasible set

$$\Omega = \{x \mid g(x) = 0,\ h(x) \leq 0\}$$

Assume f, g, h have continuous 1^{st} and 2^{nd} derivatives.

Let's derive the local optimality conditions.

LOCAL OPTIMALITY

x^* local minimizer $\iff \exists\, \delta > 0$ s.t. $f(x^*) \leq f(x) \quad \forall\, x \in \mathcal{B}_\delta(x^*) \cap \Omega$

Strict local optimality implies:

$$f(x^* + \alpha d) - f(x^*) > 0 \qquad \forall\, d,\ \exists\, \alpha > 0 \text{ s.t. } x^* + \alpha d \in \Omega$$

$$\alpha \nabla f(x^*)^T d + \frac{1}{2}\alpha^2 d^T \nabla^2 f(x^*) d + O(\alpha^3) > 0$$

This can hold only if:

$$\nabla f(x^*) = 0 \qquad \nabla^2 f(x^*) \succ 0$$

UNCONSTRAINED CASE

No direction d can locally improve the cost

UNCONSTRAINED CASE

1st-order Necessary Conditions of Optimality

x^* local optimum $\Rightarrow \nabla f(x^*) = 0$

2nd-order Sufficient Conditions of Optimality

$\nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \succ 0 \Rightarrow x^*$ strict local minimizer

$\nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \prec 0 \Rightarrow x^*$ strict local maximizer

$\nabla^2 f(x^*)$ indefinite $\Rightarrow$ Nothing can be said

EQUALITY CONSTRAINTS

Feasible cone: $F(x) = \{ d \mid \nabla g(x)^T d = 0 \}$ → Null space of linear approximation of equality constraints

Set of descent directions: $D(x) = \{ d \mid d^T \nabla f(x) < 0 \}$ → objective function decreases

Local minimum x $\Rightarrow$ $D(x) \cap F(x) = \emptyset$

Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{cases} g(x) = 0 \\ d^T \nabla f(x) \geq 0 \quad \forall d \in F(x) \end{cases}$$

→ Not practical to check every $d \in F(x)$

$\nabla g(x)^T$ = Jacobian

$d \in F(x) \iff d \in \text{null}(\nabla g(x)^T)$ → multidimensional function → More practice to check

$\Rightarrow \nabla f(x) \in \text{range}(\nabla g(x)) \iff \nabla f(x) = \nabla g(x) \lambda$

Define Lagrangian: $\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x)$

KKT conditions can be written as:

$$\nabla \mathcal{L}(x, \lambda) = 0 \iff \begin{cases} \nabla f(x) + \nabla g(x)\ \lambda = 0 & \rightarrow \text{Derivation wrt } x \\ g(x) = 0 & \rightarrow \text{Derivation wrt } \lambda \end{cases}$$

INEQUALITY CONSTRAINTS

Inequality constraints are important only at the boundaries

We can move in any direction if it is inactive. Locally you can move

Define set of active constraints $A|(x) = \{i \mid h_i(x) = 0\}$

Feasible cone $F(x) = \{d \mid \nabla g(x)^T d = 0, \nabla h_i(x)^T d \leq 0 \quad i \in A|(x)\}$

↳ considering only active constraints, by moving in direction d, active constraints stay constant or decrease

KKT CONDITIONS:

$$\begin{cases} g(x) = 0, \quad h(x) \leq 0 \\ d^T \nabla f(x) \geq 0 \quad \forall d \in F(x) \end{cases}$$

For every direction in feasible cone, my cost function must increase? or stay the same

↓ Farkas Lemma

$$\Leftrightarrow \nabla f(x) + \nabla g(x) \lambda + \nabla h_{A|}(x) \mu_{A|} = 0$$

$\nabla g(x)$: Jacobian of equality constr. ?

$\nabla h_{A|}(x)$: Jacobian of active inequality constraints

$$\mu_{A|} \geq 0$$

$$\begin{cases} \nabla f(x) + \nabla g(x)\,\lambda + \nabla h(x)\,\mu = 0 \\ \mu \geq 0 \\ \mu^T h(x) = 0 \end{cases}$$

μ → one multiplier for every constraint

$\Leftrightarrow$

$\mu^T h(x) = 0$ ← COMPLEMENTARITY CONDITION

Either $\mu_i = 0$ or $h_i(x) = 0$ or both ⇒ if a in. constr. is active, then μ_i can be positive,
if an in. constr. is inactive, the μ_i must be zero.

- if $h(x) = 0 \Rightarrow \mu_i$ can be > 0
- if $h(x) < 0 \Rightarrow \mu_i = 0$

SOLVING KKT CONDITIONS

$$\min_x \quad f(x)$$
$$\text{s.t.} \quad g(x) = 0$$
$$h(x) \leq 0$$

$\Rightarrow$ KKT system:

$$\nabla_x \mathcal{L}(x, \lambda, \mu) = 0$$
$$g(x) = 0$$
$$h(x) \leq 0$$
$$\mu \geq 0$$
$$\mu^T h(x) = 0$$

case of eq. constraints

Extra conditions for ineq. constraints

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$$

Let's focus on the case w/o inequalities.

NEWTON METHOD

$r(x) = 0$ Current guess $\bar{x}$

$$r(\bar{x} + \Delta x) \simeq r(\bar{x}) + \nabla r(\bar{x})^T \Delta x = 0$$

$$\Delta x = -\nabla r(\bar{x})^{-T} r(\bar{x})$$ ← NEWTON STEP

INVERTIBLE?

Apply Newton step:

$$x^{(k+1)} = x^{(k)} + \Delta x$$

Repeat until $\|r(x)\| <$ threshold

SOLVE KKT WITH NEWTON

$$\begin{cases} \nabla f + \nabla g \, \lambda = 0 \\ g(x) = 0 \end{cases} \iff r(x, \lambda) = \begin{bmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ g(x) \end{bmatrix} = 0$$

$$r(\bar{x} + \Delta x, \bar{\lambda} + \Delta\lambda) \simeq r(\bar{x}, \bar{\lambda}) + \nabla r^T \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = 0$$

$$\underbrace{\begin{bmatrix} \nabla^2_{xx} \mathcal{L}(\bar{x}, \bar{\lambda}) & \nabla g(\bar{x}) \\ \nabla g(\bar{x})^T & 0 \end{bmatrix}}_{\text{KKT matrix}} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -\nabla_x \mathcal{L}(\bar{x}, \bar{\lambda}) \\ -g(\bar{x}) \end{bmatrix}$$

KKT matrix

Symmetric indefinite

DESCENT DIRECTION & REGULARIZATION

Consider unconstrained problem: $\min_x f(x)$

Direction d is descent direction $\Leftrightarrow \nabla f(x)^T d < 0$

Newton step is: $\Delta x = -H^{-1} \nabla f(x)$ $\quad H \triangleq \nabla^2 f(x)$

↑ Hessian

Is Δx a descent direction?

$$-\nabla f(x)^T H^{-1} \nabla f(x) < 0 \ ?$$

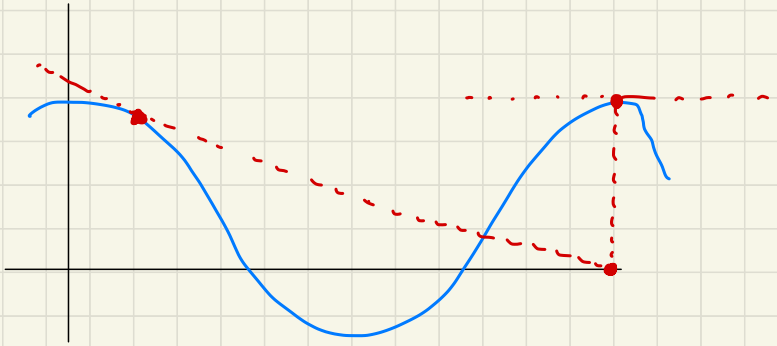
Yes if $H \succ 0$. If $\nabla^2 f(x)$ is not P.D. => regularize:

$$H = \nabla^2 f(x) + \lambda I$$

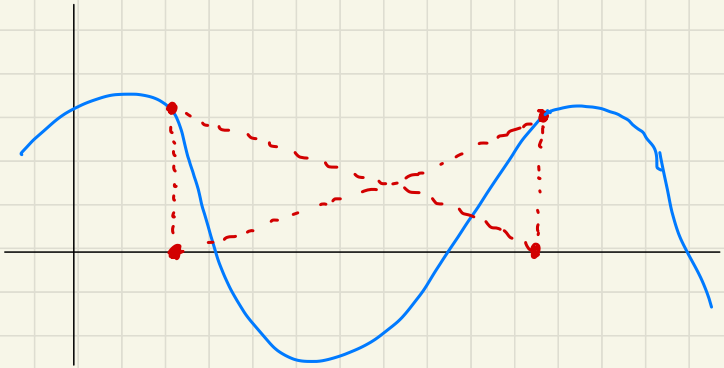
Choose $\lambda > 0$ so that $H \succ 0$

ISSUES WITH VANILLA NEWTON

① You can get stuck at points of zero slope



② You can stuck in cycles or diverge



BACKTRACKING LINE SEARCH

① Consider taking a fraction of the Newton step: $x^{(i+1)} = x^{(i)} + \alpha \Delta x$ (Step size $\alpha \in [0, 1]$)

② Observation: if $f(x)$ were linear $\underbrace{f^{(i)} - f^{(i+1)}}_{\text{reduction of cost}} = \underbrace{\alpha \nabla f_i^T \Delta x}_{\text{expected from linear approximation}}$

③ In practice $f(x)$ is nonlinear $\Rightarrow$ we'd be happy if cost is reduced by a fraction γ of what linearization predicts

GOAL: $f^{(i)} - f^{(i+1)} \geq \gamma \alpha \nabla f_i^T \Delta x$

ARMIJO'S CONDITION

Typically γ is very small, e.g., $\gamma < 10^{-2}$

PSEUDO-CODE: NEWTON STEP - UNCONSTRAINED

- INPUTS: $x^{(i)}$
- OUTPUT: $x^{(i+1)}$

Hyperparameters:
- $\beta \in [0, 1]$
- $\gamma \in [0, 1]$
- $\lambda > 0$

① $f = f(x^{(i)})$

② $\Delta x = -(\nabla r^T + \lambda I)^{-1} r$

③ $x_{try} = x^{(i)} + \alpha \Delta x, \quad \alpha = 1$

$reduction = f - f(x_{try})$

while $reduction < \gamma \alpha \nabla f^T \Delta x$

$\alpha = \beta \alpha$

$x_{try} = x^{(i)} + \alpha \Delta x$

$reduction = f - f(x_{try})$

$x^{(i+1)} = x_{try}$

LINE SEARCH WITH EQUALITY CONSTRAINTS

Two requirements on Δx:

i) reduce $f(x)$

ii) reduce $\|g(x)\|$

Introduce ℓ_1 **merit function**:

$$M_1(x) = f(x) + \sigma \|g(x)\|_1$$

Exact merit function if:

x^*, λ^* satisfy KKT conditions $\Leftrightarrow$ x^*, λ^* is local min. of $M_1(x)$

$M_1(x)$ is exact iff $\sigma > \|\lambda\|_\infty$

σ adapted at each iteration

INEQUALITY CONSTRAINTS

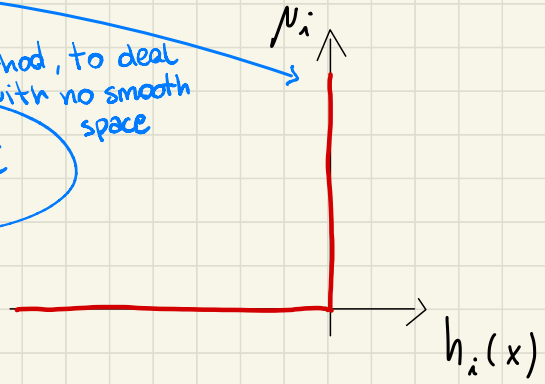
Newton method cannot be directly used because solution space generated by complementarity constraints is not smooth:

$$\mu^T h(x) = 0 \Rightarrow \mu^T h(x) \leq \varepsilon$$

Interior point method, to deal with no smooth space

$$h(x) \leq 0$$

$$\mu \geq 0$$



Two main approaches exist:

- Active set → Transform active constr. into equality constr. and remove inactive constr.
- Interior point

DIRECT METHODS

KEY IDEA: Discretize OCP ⇒ Use NLP solver to find local optimum

The finer the discretization ⇒ better approximation (+) (−)

Discretization:

① Parametrize state/control trajectories (e.g. with polynomials)

② Enforce constraints (dynamics + path inequalities) on a time grid (+)

$$t_0 < t_1 < t_2 < \ldots < t_N$$

Tons of theory on how to choose traj. parametrization and time grid so that NLP is:

i) good approximation of original OCP

ii) well conditioned → important to solve linear eq. → Newton method rely on linear eq.

DIRECT METHODS: FAMILIES

① SINGLE SHOOTING: (SEQUENTIAL)
- Discretize only $u(t)$ (trajectory)
- Compute $x(t)$ (state trajectory) integrating dynamics
- No need to have dynamics constraints

② COLLOCATION: (SIMULTANEOUS)
- Discretize both $x(t)$ and $u(t)$
- Enforce dynamics on time grid
  - ↳ To ensure that $x(t)$ and $u(t)$ are coherent with dynamics

③ MULTIPLE SHOOTING:
- Discretize all $u(t)$
- Discretize $x(t)$ at few points in time
- Compute intermediate values of x by numerical integration

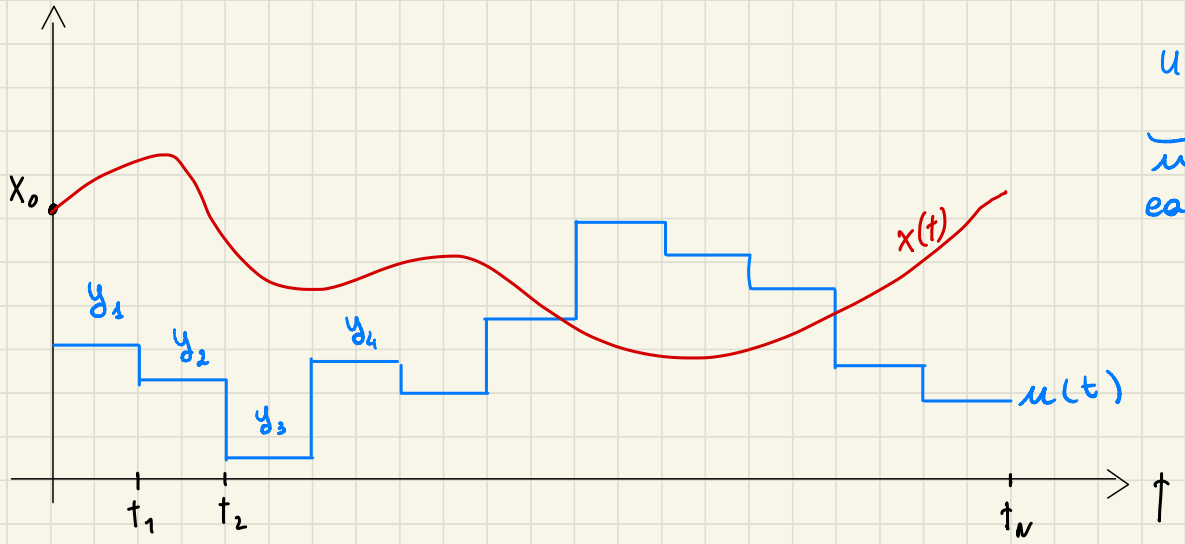
① SINGLE SHOOTING

Discretize $u(t)$ on fixed grid $0 = t_0 < t_1 < \dots < t_N = t_f$

optimization variable

$$u(t) = y_i \quad \forall t \in [t_i, t_{i+1}]$$

$u(t)$ remain cost in each interval $[t_i, t_{i+1}]$



Compute $x(t)$ from $u(t)$ and x_0 by integrating dynamics:

$$\begin{cases} \dot{x}(t) = f(x, u, t) \\ x(0) = x_0 \end{cases}$$

Initial state is fixed and known

SINGLE SHOOTING - NLP

$$\underset{y}{\text{minimize}} \quad \int_0^{t_f} l(x(t;y), u(t;y), t)\, dt + l_f(x(t_N;y))$$

Both state and control depends on y

$$\text{subject to} \quad g(x(t_i;y), u(t_i;y), t_i) \leq 0 \qquad i = 0 \dots N$$

Discretize path constraints

Running cost integral can be computed when integrating dynamics:

$$c(t) = \int_0^t l(\cdot,\cdot)\, dt \quad \Rightarrow$$

$$\begin{cases} \dot{c}(t) = l(\cdot,\cdot) \\ c(0) = 0 \end{cases}$$

$$\bar{x} = (x, c)$$

$$\dot{\bar{x}} = \begin{bmatrix} f(x,u,t) \\ l(x,u) \end{bmatrix}$$

NUMERICAL INTEGRATION

Given ODE: $\dot{x}(t) = f(x(t), t)$, $x(0) = x_0$

Compute $x(t)$ $\forall\, t \in [0, T]$.

If f is Lipshitz continuous $\Rightarrow$ unique solution x.

≈ bounded first derivatives

Neglect u because u depends just on t and x.

EXPLICIT EULER

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

↳ Definition of derivative

Take h small (h is a time interval)

$\Rightarrow$ compromise between computation and accuracy:

smaller h $\Rightarrow$ ↑comp. ↑accuracy
bigger h $\Rightarrow$ ↓comp. ↓accuracy

$$h\, f(x,t) \simeq x(t+h) - x(t) \Rightarrow x(t+h) = x(t) + h\, f(x,t)$$

Ok if h is sufficiently small $\Rightarrow$ Computationally expansive!

Not very accurate (order 1)

RUNGE-KUTTA METHODS

$$x_{n+1} = x_n + h \sum_{i=1}^{q} b_i k_i$$

{ Approximate average $\dot{x}$ with weighted average of k_i, with weights b_i, with $\sum_i b_i = 1$, $b_i \geq 0$

$$k_i = f\left(x_n + h \sum_{j=1}^{q} a_{ij} k_j,\ t_n + c_i h\right)$$

{ Approximate value of $\dot{x}$ at time $t_n + c_i h$, with $c_i \in [0, 1]$

a_{ij}, c_i: coefficients that say when and where evaluate dynamics

b_i: how many evaluations of dynamics to compute

- q is the order of the method
- For $q = 1$ we recover Euler ($b_1 = 1$, $a_{11} = 0$, $c_1 = 0$)
- If $a_{ij} = 0 \quad \forall j \geq i$ => Explicit method

  otherwise => Implicit method => need to solve system of nonlinear equations

  ↑ more computationally demanding, but more stable

- For $q > 1$ there exists many versions of same order p:

  ↑ (consistency)

$$\lim_{h \to 0} \left[x(t) - \hat{x}(t; t-h, x(t-h)) \right] = O(h^{p+1})$$

Exact trajectory — Integrator output

BUTCHER TABLEAU

$$\begin{array}{c|c} c_1 & a_{11} \cdots a_{1q} \\ \vdots & \vdots \\ c_q & a_{q1} \cdots a_{qq} \\ \hline & b_1 \cdots b_q \end{array}$$

- Uniquely defines an RK method
- Most common RK4 method is:

$$k_1 = f(x_n, t_n)$$

$$k_2 = f\left(x_n + \tfrac{1}{2} h k_1, t_n + \tfrac{h}{2}\right)$$

$$k_3 = f\left(x_n + \tfrac{1}{2} h k_2, t_n + \tfrac{h}{2}\right)$$

$$k_4 = f(x_n + h k_3, t_n + h)$$

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

EXAMPLE OF METHOD FOR ORDER 4

order 4 →

If they are non-zero you have an implicit method

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Evaluate dynamics 4 times for every step of integration

⇓

very accurate but very comp. demanding

For $p \geq 5$ ∄ method with $p = q$! → with order 5 we have to do 6 evalu. of dynamics and so on

SUMMARY - SINGLE SHOOTING

- Remove x from decision variables
- Discretize $u(t) = y_i \quad \forall t \in [t_i, t_{i+1}]$
- Compute x by integrating dynamics $\dot{x} = f(x, u, t)$
- Remove dynamics from constraints

I recover x integrating

- High-order integration schemes (typically) more efficient
- Differentiation of integration scheme often computed using automatic diffentiation libraries.
- Implicit schemes typically needed for stiff dynamics

② COLLOCATION

Discretize $x(t)$ with polynomials (e.g. order $k=0$) on fine grid:

$$x(t) = s_i \qquad \forall\, t \in [t_i, t_{i+1}]$$

One polynomial for each time interval

Replace dynamics $\dot{x} = f(x, u)$ with:

I approximate $\dot{x}$

Appr. of dynamic: $\frac{s_{i+1} - s_i}{t_{i+1} - t_i}$ — Dynamics: $f(s_i, y_i) = 0$

$i = 0 \dots N-1$

$\Rightarrow$ Appox. of dynamics must be equal to dynamics

$\Downarrow$

Best approximation possible

$$c_i(s_i, s_{i+1}, y_i) = 0$$

Approximate cost integral:

$$\int_{t_i}^{t_{i+1}} l(x(t), u(t))\, dt \simeq l(s_i, y_i)(t_{i+1} - t_i) \triangleq l_i(s_i, y_i)$$

COLLOCATION NLP (SPARSE)

$$\underset{s,\,y}{\text{minimize}} \quad \sum_{i=0}^{N-1} l_i(s_i, y_i) + l_f(s_N)$$

$$\text{subject to} \quad s_0 - x_0 = 0$$

⇒ opt. problem

← initial conditions

discretized dynamics → $c_i(s_i, s_{i+1}, y_i) = 0 \quad i = 0 \dots N-1$

discretized path constraints → $g_i(s_i, y_i) \leq 0$ only check them $i = 0 \dots N-1$ at discrete time instants

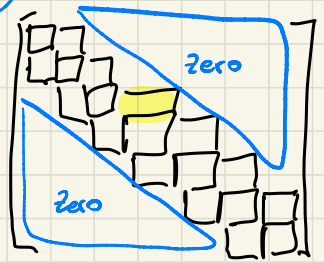
Why sparse? Because KKT matrix is sparse!

↑ Most of the entry of the matrix are zero

computation became more efficient

$$H_{ij} = \frac{\partial}{\partial s_i}\left(\frac{\partial}{\partial s_j} L\right) = 0 \quad \forall\, i \neq \{j, j+1, j-1\}$$

$H =$



③ MULTIPLE SHOOTING

Discretize control $u(t)$ on coarse grid $t_0 < t_1 < \dots < t_N$

$$u(t) = y_i \quad \forall t \in [t_i, t_{i+1}]$$

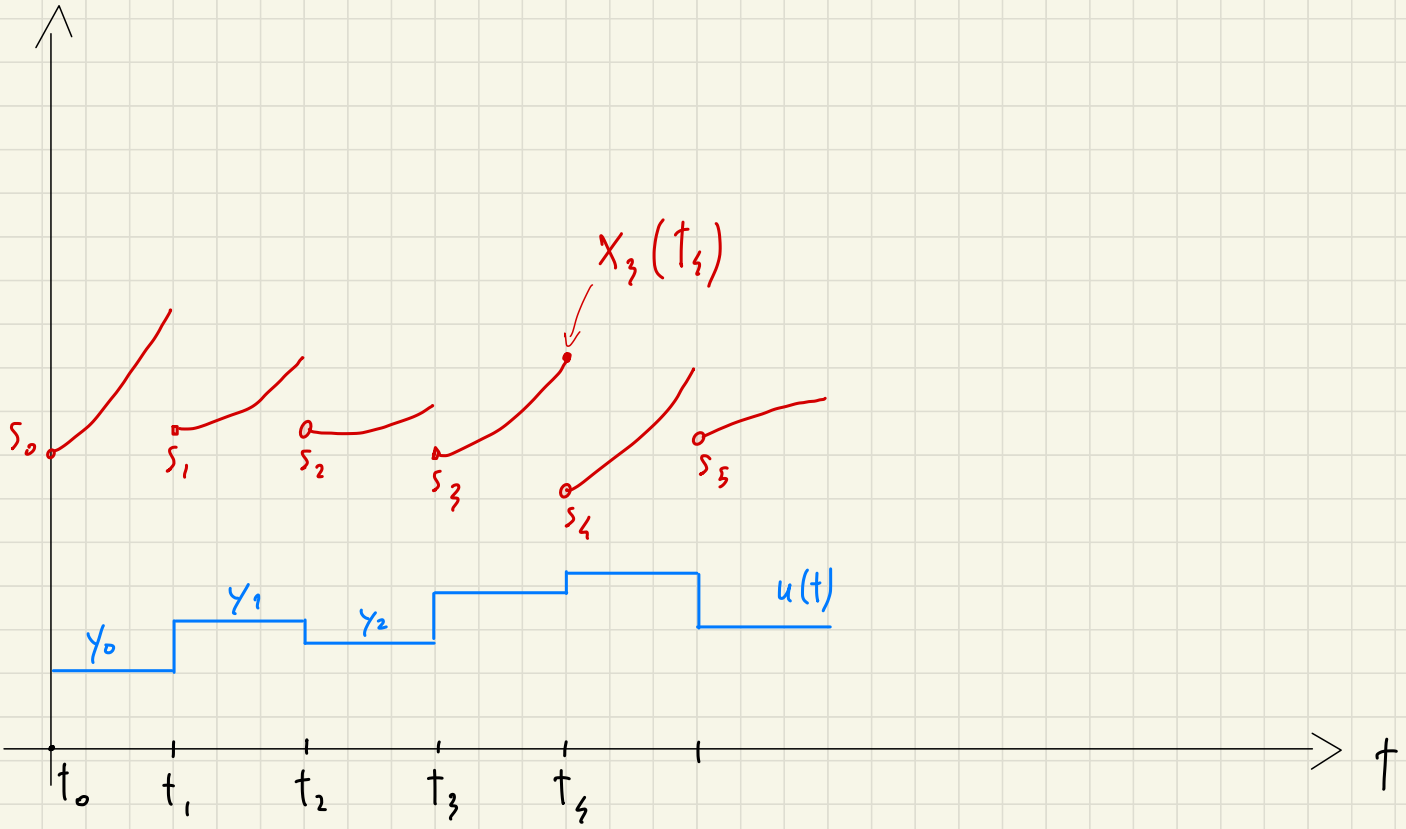
Integrate numerically ODE in each interval $[t_i, t_{i+1}]$ state variable s_i:

$$\dot{x}(t) = f(x_i(t), y_i) \qquad t \in [t_i, t_{i+1}]$$
$$x_i(t_i) = s_i$$

Obtain $x_i(t)$. Numerically compute integrals:

$$l_i(s_i, y_i) = \int_{t_i}^{t_{i+1}} l(x_i(t), y_i)\, dt$$

MULTIPLE SHOOTING SKETCH



MULTIPLE SHOOTING NLP

$$\underset{s,y}{\text{minimize}} \quad \sum_{i=0}^{N-1} l_i(s_i, y_i) + l_f(x_N)$$

subject to

$$s_0 - x_0 = 0 \qquad \text{initial conditions}$$

$$s_{i+1} - \underbrace{x_i(t_{i+1}; s_i, y_i)}_{\text{Result of numerical integration from } s_i} = 0 \qquad \text{continuity constr.}$$

$$g_i(s_i, y_i) \leq 0 \qquad \text{path constraints}$$

DIRECT METHODS - SUMMARY

① SINGLE SHOOTING

⊕ Small problem

⊕ Need initial guess only for $u(t)$

⊕ Can use off-the-shelf ODE solvers

⊖ Cannot exploit knowledge of $x(t)$ in initialization

⊖ Numerical issues for unstable systems

③ MULTIPLE SHOOTING

• Medium problem

⊕ Sparser than ①

⊖ Less sparse than ②

⊕ Unstable OK

⊕ Initialize $x(t)$

⊕ Can adapt time grid

② COLLOCATION

⊖ Large problem

⊖ Cannot adapt time grid

⊕ Sparse problem

⊕ Work well with unstable systems

⊕ Can initialize $x(t)$

MODEL PREDICTIVE CONTROL

- Use optimal control for controlling system (robot)
- Solve a finite-horizon OCP using current state as initial state

$$X^*, U^* = \underset{X,U}{\arg\min} \sum_{i=0}^{N-1} l(x_i, u_i)$$

subject to

$$x_{i+1} = f(x_n, u_n, k) \quad i = 0 \dots N-1$$

(limits on state-var.)

$$x_{i+1} \in \mathcal{X}, \quad u_i \in \mathcal{U} \quad i = 0 \dots N-1$$

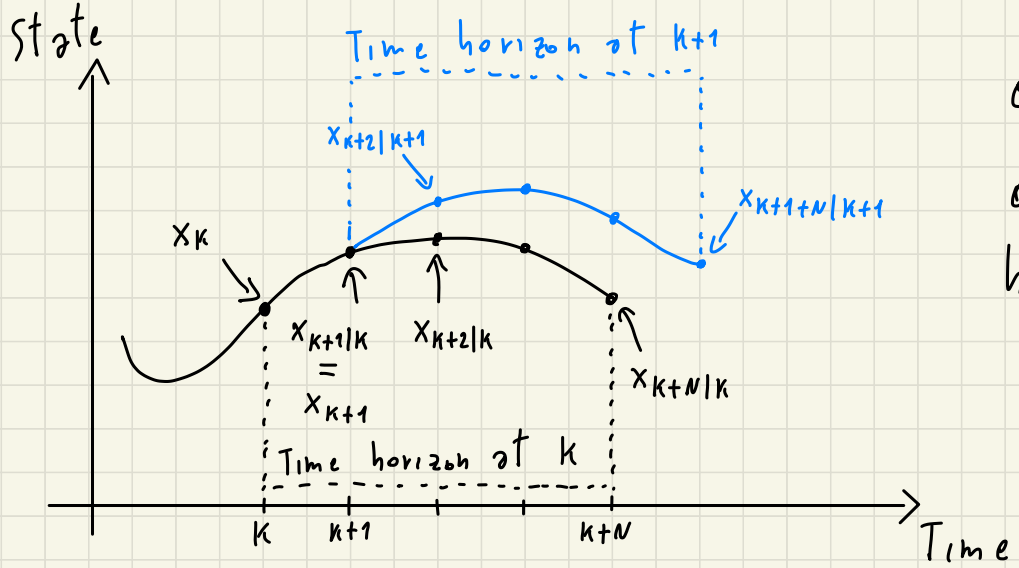
(limits on control)

$$x_0 = x^{meas}$$

- Apply computed control for this time step: $\pi = u_0^*$
- Do not use the rest of U^*, nor X^*
- Repeat the same process at next control loop → to keep being updated with environment

Remain optimal despite to changes in environment

PREDICTED VS ACTUAL TRAJECTORIES



OCP at time k and k+1 optimize over different horizons so can result in different trajectories

Even without disturbances, predicted and actual trajectories can be different!

CHALLEGES IN MPC

① FEASIBILITY

- Can I ensure the OCP is always feasible?
- What can I do if it's not feasible?

② STABILITY

- Can I ensure MPC stabilizes the system?

③ COMPUTATION TIME

- Can I solve the OCP sufficiently fast?

INFINITE-HORIZON MPC

- If $N = \infty$ the time horizon seen by the solved OCP's is the same
- In this case predicted and actual trajectories are the same (assuming no disturbances)
- This follows from Bellman's optimality principle:

> Given optimal traj. $x^*(0) \dots x^*(N)$, the subtraj. $x^*(k) \dots x^*(N)$ is optimal for OCP on horizon $[k, N]$ starting from $x(k)$

INFINITE HORIZON STABILITY & FEASIBILITY

With $N = \infty$, if cost $\ell(x, u) \geq \alpha \|x\|$ $\forall x, u$ for some $\alpha > 0$.
Then having a finite cost implies **stability**.

Moreover, since predicted and actual traj. are the same, feasibility of the first OCP implies feasibility of all successive OCP's (**recursive feasibility**)

IDEA:

Add **terminal cost** and constraints to finite-horizon OCP to mimic an infinite horizon.

Reminiscent of **VALUE FUNCTION**

TERMINAL COST & CONSTRAINTS

$$V_N^*(\bar{x}) = \underset{U}{\text{minimize}} \sum_{k=0}^{N-1} \ell(x_k, u_k) + \ell_f(x_N)$$

TERMINAL COST

subject to

$$x_{k+1} = f(x_k, u_k) \qquad k = 0 \dots N-1$$

$$x_{k+1} \in \mathcal{X}, \quad u_k \in \mathcal{U} \qquad k = 0 \dots N-1$$

$$x_0 = \bar{x}$$

$$x_N \in \mathcal{X}_f$$

TERMINAL CONSTRAINT

How do we choose $\ell_f(\cdot)$, $\mathcal{X}_f$?

FEASIBILITY

FEASIBILITY

Is the OCP going to be feasible at all sampling instants t?

- INPUT CONSTRAINTS ONLY => Always feasible
- HARD STATE CONSTRAINTS => If $N < \infty$ there is no guarantee that OCP remains feasible, even in the nominal case.

Maximum output admissible set theory:

- $N < \infty$ is enough
- Hard to know which value of N is enough
- Computation time may be too high for such N

THEOREM

If $\mathcal{X}_f$ is control-invariant $\Rightarrow$ MPC is recursively feasible

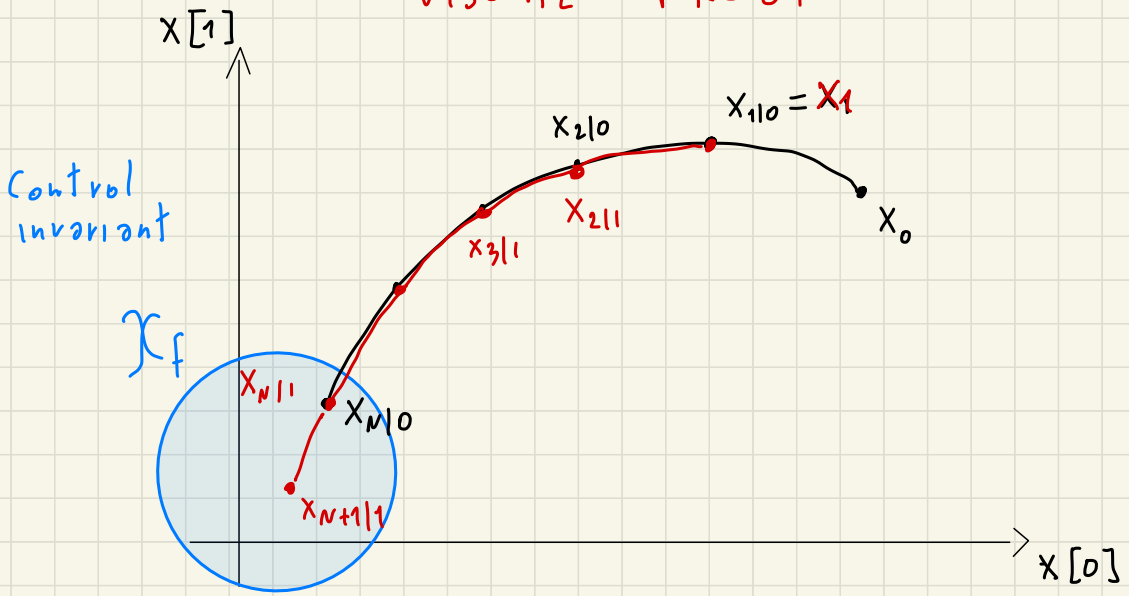
---

DEFINITION

- A set S is control invariant if $\forall x \in S$, $\exists u \in \mathcal{U}$ s.t. $f(x,u) \in S$

i.e. if you start in S, you can remain in S.

VISUAL PROOF



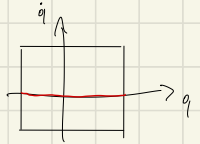
$\mathcal{X}_f \subseteq \mathcal{X}$

CONTROL INVARIANT SETS

Example: for a manipulator, set of zero velocity states is control invariant if

$$|g(q)| \leq \tau^{max} \quad \forall q \in [q^{min}, q^{max}]$$

$$S = \{(q, 0) \mid q^{min} \leq q \leq q^{max}, \; q \in \mathbb{R}^n\}$$



How do I compute C.I. sets in general?

- Hard problem for nonlinear systems
- Set of equilibria S is always C.I., but small:

$$S = \{x \in X \mid \exists u \in U : \; x = f(x, u)\}$$

- Can use backward reachable set of S, i.e. states starting from which you can reach S

→ Aggiungi schema

COMPUTING N-STEP BACKWARD REACHABLE SETS

- We can easily check whether a state $\bar{x} \in \mathcal{B}_N(S)$ (N-step backward reachable set of S) by solving this OCP:

$$
\begin{aligned}
&\underset{x,u}{\text{minimize}} && 1 && \\
&\text{subject to} && x_{i+1} = f(x_i, u_i) && i = 0 \dots N \\
& && x_{i+1} \in \mathcal{X}, \quad u_i \in \mathcal{U} && i = 0 \dots N \\
& && x_N = x_{N+1} && \\
& && x_0 = \bar{x} &&
\end{aligned}
\tag{1}
$$

$x_N \in S \quad \longleftrightarrow$

- If a solution exists $\Rightarrow \bar{x} \in \mathcal{B}_N(S)$
- Otherwise $\Rightarrow \bar{x} \notin \mathcal{B}_N(S)$

- Use function approximation techniques to learn an approximate representation of $\mathcal{B}_N(S)$ from examples
- To build dataset, repeat the following steps:
  - Sample state $\bar{x}$
  - If $\bar{x} \notin X$ => Add data point $(\bar{x}, 0)$ to dataset
  - Else, try to solve OCP (1)
  - If solution found => Add $(\bar{x}, 1)$ to dataset
  - Else => Add $(\bar{x}, 0)$ to dataset
- When dataset is built, train classifier (e.g. Neural network) to predict label (0 or 1) given state.
- Finally, use classifier as terminal constraint in MPC

FINE

STABILITY

LYAPUNOV FUNCTIONS

Def: $V: \mathbb{R}^n \to \mathbb{R}$ is **exponential Lyapunov function** if $\exists\, \alpha_1, \alpha_2, \alpha_3 > 0$ s.t. $\forall x \in \mathcal{X}$:

$$\alpha_1 |x| \leq V(x) \leq \alpha_2 |x|$$

$\Rightarrow V(x) > 0 \quad \forall x \neq 0$

$\Rightarrow V(0) = 0$

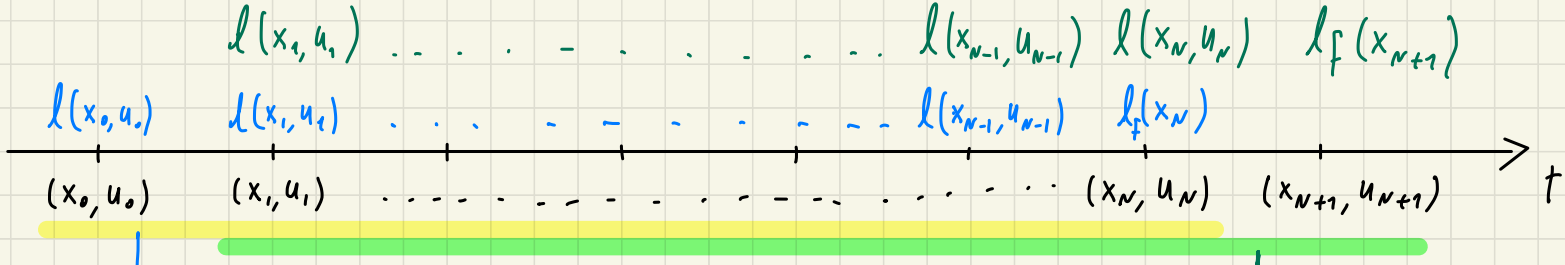
$$V(f(x)) - V(x) \leq -\alpha_3 |x|$$

$\Rightarrow V(x_1) \leq V(x_0) - \alpha_3(x_0)$

Th: If $\exists$ Lyap. func. $\Rightarrow$ origin ($x = 0$) is **exponentially stable** in $\mathcal{X}$.

$$\|x_k\| \leq c\, \gamma^k \|x_0\| \quad \text{for some } c > 0,\ \gamma \in [0,1]$$

VALUE FUNCTION AS LYAPUNOV FUNCTION



$$V_0^*(x_0) = \sum_{i=0}^{N-1} l(x_i,u_i) + l_f(x_N)$$

$$V_1(x_1) = \sum_{i=1}^{N} l(x_i,u_i) + l_f(x_{N+1})$$

$$V_1(x_1) - V_0^*(x_0) = l(x_N,u_N) + l_f(x_{N+1}) - l_f(x_N) - l(x_0,u_0) \leq -\alpha_3|x_0| \quad \forall x_N \in \mathcal{X}_f$$

Sufficient but not necessary because $V_1(x_1)$ is not optimal!

STABILITY ASSUMPTIONS

Ass #1: $f(0,0)=0, \quad l(0,0)=0, \quad l_f(0)=0$

Ass. #2: $\forall x \in X_f, \exists u \in U$ s.t.:

$$l_f(f(x,u)) - l_f(x) \leq -l(x,u)$$

Terminal cost decrease

Ass. #3: $\exists \alpha_1 > 0$ s.t. $\forall x \in X_N, \forall u \in U$

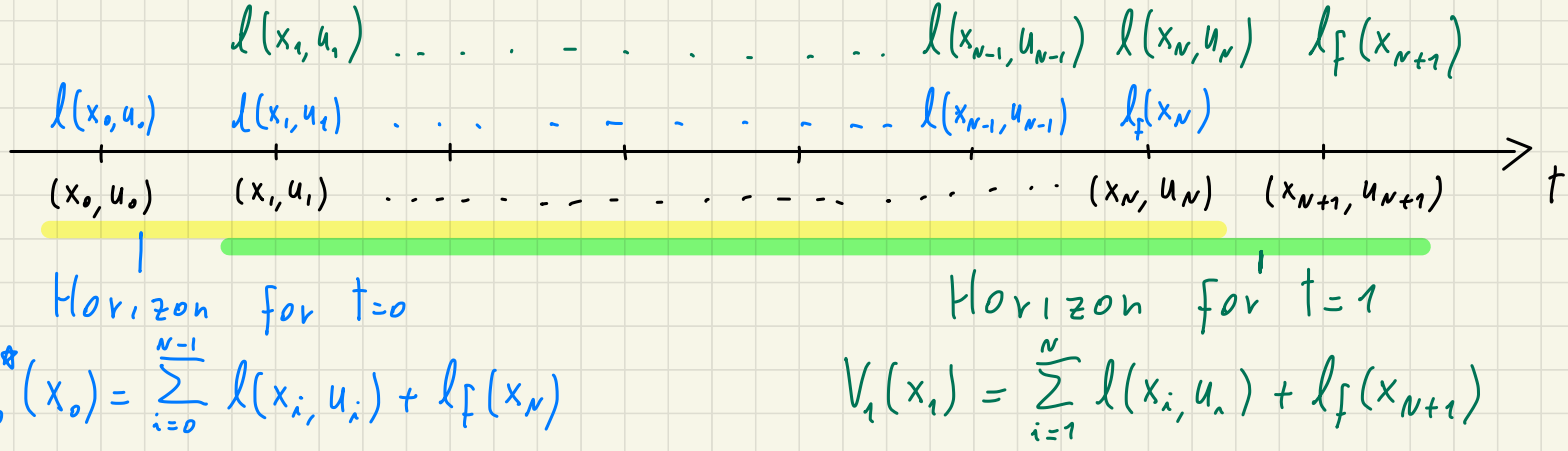
X_N: Set of states from which O.C.P. has a solution

$$l(x,u) \geq \alpha_1 |x|$$

Cost lower bound

If #1, #2, #3 hold $\Rightarrow$ $V^*(\cdot)$ is an exp. Lyapunov function

VALUE FUNCTION AS LYAPUNOV FUNCTION



$$V_0^*(x_0) = \sum_{i=0}^{N-1} \ell(x_i,u_i) + \ell_f(x_N)$$

$$V_1(x_1) = \sum_{i=1}^{N} \ell(x_i,u_i) + \ell_f(x_{N+1})$$

$$V_1(x_1) - V_0^*(x_0) = \underbrace{\ell(x_N,u_N) + \ell_f(x_{N+1}) - \ell_f(x_N)}_{\leq 0 \text{ by Ass. \#2}} \underbrace{- \ell(x_0,u_0)}_{\leq -\alpha_1|x_0| \text{ by Ass. \#3}} \leq -\alpha_3|x_0| \qquad \forall\, x_N \in \mathcal{X}_f$$

STABILITY ASSUMPTIONS

Note that assuming $f(0,0)=0$ is not restrictive. If we want to stabilize $x^* \neq 0$, s.t. $f(x^*, u^*) = x^*$, i.e. x^* must be an equilibrium $\Rightarrow$ Define new state $\bar{x} = x - x^*$, control $\bar{u} = u - u^*$, dynamics:

$$\left.\begin{aligned} x^+ &= f(x,u) \\ x^* &= f(x^*, u^*) \end{aligned}\right\} \quad \begin{aligned} x^+ - x^* &= f(\bar{x} + x^*, \bar{u} + u^*) - x^* \\ \bar{x}^+ &= \bar{f}(\bar{x}, \bar{u}) \end{aligned}$$

So that $\bar{f}(0,0) = f(0+x^*, 0+u^*) - x^* = 0$

STABILITY OPTIONS

① Pick $\mathcal{X}_f$ and l_f (e.g. using sampling-based approach and function approximators) such that:

- $\mathcal{X}_f$ is control invariant
- $\forall x \in \mathcal{X}_f \quad \exists u \in \mathcal{U}$ s.t. $l_f(f(x,u)) - l_f(x) \leq -l(x,u)$

② Pick $\mathcal{X}_f = \{0\}$ $\Rightarrow$ Small basin of attraction

② is special case of ①

③ Pick N "large enough" and set $l_f(x) = 0$, $\mathcal{X}_f = \mathcal{X}$

↑ Most common in practice, especially for nonlinear systems!

Downside: large N $\Rightarrow$ large computation time!

STABILITY EXTENSIONS

- What if system is time varying?
  - e.g., trajectory tracking can be cast as regulation of time-varying system

- Stability + recursive feasibility can still be ensured with time-varying $\mathcal{X}_f$, $l_f(\cdot)$

- What if cost measures some kind of energy comsumption
  => does not satisfy $l(x,u) \geq \alpha|x|$

  => Economic MPC

③ COMPUTATION TIME

- KEY IDEA: **warm start**, i.e. use solution of previous OCP as initial guess for current OCP
- Shift trajectory back by 1 time step: $u_n^{guess} \leftarrow u_{n+1}^{\star}$
- Use zero as initial guess for last time step: $u_{N-1}^{guess} \leftarrow 0$
- **Don't iterate until convergence**, just do 1 Newton step
  - Possibly skip line search
- Use hardware accelerators (e.g. GPU)

REINFORCEMENT LEARNING

- Born in Computer Science community
- Many similarities with Optimal Control, but RL
  - tries to find **globally optimal policy**
  - assumes **dynamics is unknown** — MAIN DIFFERENCE
  - initially focused on **finite size state/control spaces**
  - uses different terminology → same concepts but with different names
  - typically assumes dynamics is **stochastic** → Friction can change while using robot
    → We will not in this course, to simplify equations
  - typically uses **infinite horizon**

TERMINOLOGY

RL	OC
State s	State x
Action a	Control u
Environment	Plant
Reward	Cost
Return	Cost to go
Maximize	Minimize
Value function	Cost to go (of a policy)
Optimal Value function	Value function

MARKOV DECISION PROCESSES (MDP)

- Describe environment for RL problem
- Fully observable
- Markov property: Future is independent of the past given the present

$$\mathbb{P}(X_{t+1} \mid X_t) = \mathbb{P}(X_{t+1} \mid X_1, \dots, X_t)$$

- State transition probability: $\mathcal{P}_{xx'} = \mathbb{P}(X_{t+1} = x' \mid X_t = x)$

- State transition matrix:
  - Each row sums to 1 (total law probability)
  - In deterministic systems $\mathcal{P}$ contains only 0 and 1

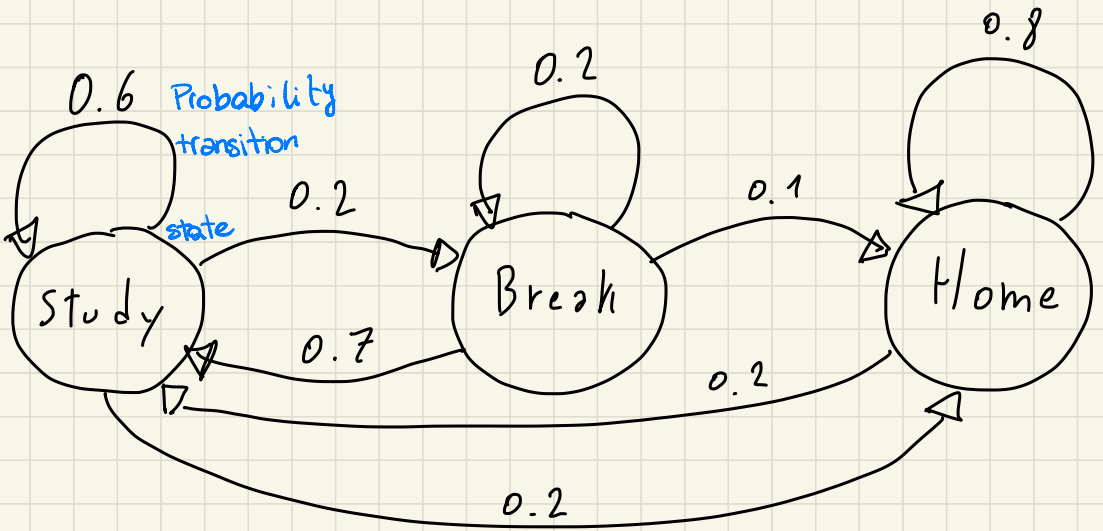
$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix}$$

MARKOV PROCESS

- Tuple $\langle X, P \rangle$
  - X = (finite) set of states
  - P = state transition probability matrix
- A.K.A. Markov Chain

	S	B	H
S	0.6	0.2	0.2
B	0.7	0.2	0.1
H	0.2	0	0.8

$= P$



STATE TRANSITION PROBABILITY MATRIX

- Since P has all nonnegative entries, and it's irreducible (Associated to a strongly connected graph) by Perron-Frobenius theorem we know that:

PROPERTY 1:

- The largest (in norm) eigenvalue r of P satisfies:

$$\min_i \sum_j P_{ij} \leq r \leq \max_i \sum_j P_{ij} \Rightarrow r = 1$$

PROPERTY 2:

- All eigenvalues of P are smaller than 1: (Directly consequence of Property 1.)

$$\lambda_i(P) < 1 \quad \forall i$$

IMPORTANT

$\|Pv\| \leq \|v\|$ → P v can only restrict the vector

PROBABILY MATRIX vs DYNAMICS FUNCTION

In OCP for deterministic systems dynamics was encoded as:

CONTINUOUS OR DISCRETE STATE SPACE ← $x^{+} = f(x)$ ← PLUS

$x^{+} = f(x, w) \quad w \sim P$ → uncertainty that follow an uncertainty prob. (usually known)

↳ uncertainty

In MP dynamics is encoded as probability density func.:

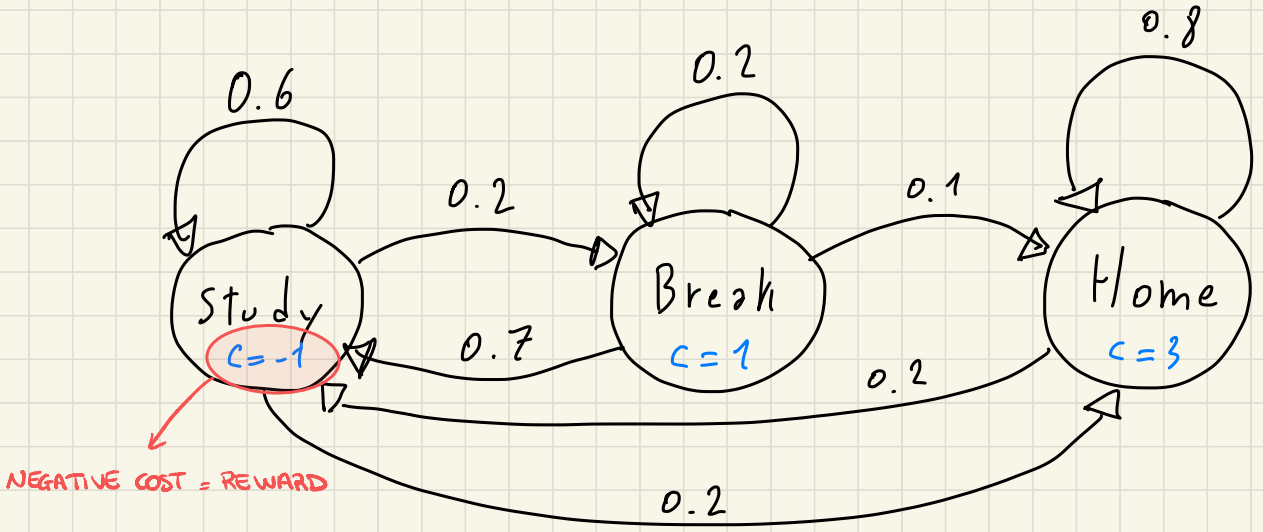
ONLY WITH DISCRETE STATE SPACE ← $\mathbb{P}(x^{+} \mid x)$ Probability of transitioning from state x to next state x^{+}

If system is deterministic, both approaches can be used.

We will favor the first one whenever possible.

MARKOV REWARD PROCESS

- Tuple $\langle \mathcal{X}, P, C, \gamma \rangle$ (state space, prob. matrix)
  - C = cost function $C_x = \mathbb{E}[\ell_{t+1} \mid x_t = x]$ (expectation since dynamic is stochastic)
  - γ = discount factor $\gamma \in [0, 1]$ PART OF PROBLEM DEFINITION
    - ↳ since we work with ∞ horizon, it prevents cost to go to ∞



COST TO GO (RETURN)

- Total discounted cost from time t to ∞:

$$J_t = \underbrace{\ell_t}_{\text{cost}} + \gamma \underbrace{\ell_{t+1}}_{\text{cost of next state}} + \ldots = \sum_{k=0}^{\infty} \gamma^k \ell_{t+k}$$

IF $\gamma \to 1$ present and future are almost the same

- γ represents preference for later costs over immediate costs
  - γ close to 0 => myopic evaluation
  - γ close to 1 => far-sighted evaluation

- Why discount?
  - Avoid infinite cost
  - Uncertainty about the future
  - Immediate cost could require interest

↑↑ γ ↓↓ convergence

γ = discount factor

$0 \leq \gamma \leq 1$

VALUE FUNCTION (COST-TO-GO)

- Cost starting from state x:

$$V(x) = J_t(x_t = x) = c_t + \gamma c_{t+1} + \gamma^2 c_{t+2} + \dots$$

$$\stackrel{!}{=} c_t + \gamma \underbrace{(c_{t+1} + \gamma c_{t+2} + \dots)}_{V(x_{t+1})}$$

- Can be decomposed in two parts:

$$V(x) = \ell(x) + \gamma V(f(x))$$

same thing written in different ways

- This is the **Bellman Equation**

- Can be expressed in matrix form

$$V = C + \gamma P V$$

→ Method to compute the value of next state $V(F(x))$

$$\begin{bmatrix} V(1) \\ \vdots \\ V(h) \end{bmatrix} = \begin{bmatrix} \ell(1) \\ \vdots \\ \ell(h) \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1h} \\ \vdots & & \vdots \\ P_{h1} & \cdots & P_{hh} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(h) \end{bmatrix}$$

→ Compute the expected value to go from state 1 To other n states

→ Weighted average in which weight is the probability

- Linear equation in V: $V = (I - \gamma P)^{-1} C$
- Direct solution only possible for small MRP's
- Iterative methods used for large MRP's

MARKOV DECISION PROCESS (MDP)

- Tuple $\langle \mathcal{X}, \mathcal{U}, \mathcal{P}, C, \gamma \rangle$ — we introduce control input (Action a)
- $\mathcal{U}$ = finite set of control inputs
- $\mathcal{P}^{u}_{xx'} = \mathbb{P}(x_{t+1} = x' \mid x_t = x, u_t = u)$
- $C^{u}_{x} = \ell_t \ (x_t = x, u_t = u)$

} cost and prob. depend also on control input

- **Policy** = distribution over control inputs given states: (acts like a controller)
  - ↳ stochastic map from state to control

$$\pi(u \mid x) = \mathbb{P}(u_t = u \mid x_t = x)$$

- We will assume **deterministic policies**: $u = \pi(x)$

We can use stochastic policy during policy search process

ACTION-VALUE FUNCTION

Cost starting from state x, applying input u, and then following policy π

$$Q^{\pi}(x,u) = J_t(x_t = x,\ u_t = u,\ u_{k>t} \sim \pi)$$

Action and policy are independent

cost starting from a given state, applying a given action

You don't know the future cost, it depends on the next action you decide to take

Can be decomposed as:

$$Q^{\pi}(x,u) = l(x,u) + \gamma Q^{\pi}(f(x,u), \pi(f(x,u)))$$

π evaluated at the next state

$$= l(x,u) + \gamma V^{\pi}(f(x,u))$$

where:

$$V^{\pi}(x) = Q^{\pi}(x, \pi(x))$$

first action it's the only that doesn't come from action set π

Relation between value function and action-value function

OPTIMAL VALUE FUNCTION

Policy that minimize the return

- Minimum Value function over all policies:

$$V^*(x) = \min_{\pi} V^{\pi}(x)$$

- Optimal action-value function:

$$Q^*(x,u) = \min_{\pi} Q^{\pi}(x,u)$$

- Optimal policy:

$$\pi^* \leq \pi \quad \forall \pi$$

[where $\pi \leq \pi'$ if $V^{\pi}(x) \leq V^{\pi'}(x) \; \forall x$]

OPTIMAL POLICY

- Minimize Q^* over u:

$$\pi^*(x) = \arg\min_{z \in V} Q^*(x, z)$$

- If we know $Q^*(x, u)$ we immediatly have optimal policy

$$\pi^*(x) = \min_u \ell(x, u) + \gamma V(f(x, u))$$

BELLMAN OPTIMALITY EQUATIONS

$$V^*(x) = \min_u \; Q^*(x,u)$$

$$Q^*(x,u) = \ell(x,u) + \gamma V^*(f(x,u))$$

$$V^*(x) = \min_u \; \ell(x,u) + \gamma V^*(f(x,u))$$

$$Q^*(x,u) = \ell(x,u) + \gamma \min_{u'} Q^*(f(x,u),u')$$

- Nonlinear
- No closed-form solution
- Iterative algorithms

DYNAMIC PROGRAMMING

- Assume full knowledge of an MDP
- Two classes of problems:

① PREDICTION:
- Input: MDP $\langle \mathcal{X}, \mathcal{U}, \mathcal{P}, C, \gamma \rangle$ and policy π
- Output: Value function V^{π}

② CONTROL:
- Input: MDP
- Output: Optimal value function V^{*} and Optimal policy π^{*}

FINITE VS INFINITE HORIZON

- **Finite** horizon:
  - theory is simpler
  - mostly used in **optimal control**
  - policy and Value depend on time

- **Infinite** horizon:
  - theory is more complex, but **elegant**
  - mostly used in **RL**
  - policy and Value are stationary
  - good approximation of problems with finite but long horizon

BELLMAN OPERATORS

- Bellman (expectation backup) operator T^π:

$$T^\pi(V) = C^\pi + \gamma P^\pi V$$

- Bellman optimality (backup) operator T:

$$T(V) = \min_{u \in \mathcal{U}} C^u + \gamma P^u V$$

Different Notation →

$$(TV)(x) = \min_{u \in \mathcal{U}} \ell(x,u) + \gamma V(f(x,u))$$

PREDICTION: ITERATIVE POLICY EVALUATION

- Problem: evaluate a given policy π
- Solution: apply iteratively Bellman expectation operator:

$$V_{k+1}(x) = \ell(x, \pi(x)) + \gamma V_k(f(x, \pi(x))) \qquad \forall x \in \mathcal{X}$$

- In short: $V_{k+1} = T^{\pi} V_k$
- Start from arbitrary Value function V_0
- Guarantee to converge to V_π: $\lim_{k \to \infty} V_k = V_\pi$

CONVERGENCE PROOF OF POLICY EVALUATION

$$V_{k+1} = T^{\pi} V_k$$

Prove T^{π} is contracting

$$\| T^{\pi} V - T^{\pi} Z \|_\infty = \| \cancel{C^{\pi}} + \gamma P^{\pi} V - \cancel{C^{\pi}} - \gamma P^{\pi} Z \|_\infty = \gamma \| P^{\pi} (V - Z) \|_\infty$$

$$\leq \gamma \| V - Z \|_\infty$$

↑ Because each row of P^{π} sums to 1

Given V_0 and V_π show $V_k \to V_\pi$ as $k \to \infty$:

$$\| V_k - V_\pi \|_\infty = \| T^{\pi} V_{k-1} - T^{\pi} V_\pi \|_\infty \leq \gamma \| V_{k-1} - V_\pi \|_\infty \leq \gamma^2 \| V_{k-2} - V_\pi \|_\infty$$

$$\leq \dots \leq \gamma^k \| V_0 - V_\pi \|_\infty$$

V_k converges to V_π at geometric rate.

Prove V_π is unique fixed point of T^{π}. Assume V and Z are both fixed points of T^{π}:

$$T^{\pi} V = V \qquad T^{\pi} Z = Z$$

Prove $V = Z$:

$$\| T^{\pi} V - T^{\pi} Z \|_\infty \leq \gamma \| V - Z \|_\infty \leftarrow \text{Because } T^{\pi} \text{ is contracting}$$

$$\| T^{\pi} V - T^{\pi} Z \|_\infty = \| V - Z \|_\infty \leftarrow \text{Because } V, Z \text{ are fixed points}$$

$$\gamma \| V - Z \|_\infty \geq \| V - Z \|_\infty \implies \| V - Z \|_\infty = 0 \quad \text{Because } \gamma < 1$$

CONTROL: POLICY ITERATION

- Problem: Find optimal policy π^*
- Solution:
  - Start with arbitrary policy π_0
  - For $k = 0 \dots \infty$
    - Evaluate $\pi_k \Rightarrow V^{\pi_k}$
    - Improve policy acting greedily w.r.t. V^{π_k}:

$$\pi_{k+1}(x) = \arg\min_u \ell(x,u) + \gamma V^{\pi_k}(f(x,u))$$

- Always converges to π^*: $\lim_{k \to \infty} \pi_k = \pi^*$

CONVERGENCE OF POLICY ITERATION

Show that $\pi_{k+1} \leq \pi_k \quad \forall k \iff \pi' \leq \pi$

$$\pi'(x) = \arg\min_u \left[l(x,u) + \gamma V^{\pi}(f(x,u)) \right]$$

$$\underbrace{\min_u \left[l(x,u) + \gamma V^{\pi}(f(x,u)) \right]}_{\text{Cost we get by following } \pi' \text{ for first step and then following } \pi} \leq \underbrace{l(x,\pi(x)) + \gamma V^{\pi}(f(x,\pi(x)))}_{\text{Cost following } \pi}$$

$$Q^{\pi}(x,\pi') = \min_u \left[Q^{\pi}(x,u) \right] \leq Q^{\pi}(x,\pi(x)) = V^{\pi}(x)$$

$$V^{\pi}(x) \geqslant \min_u \left[l(x,u) + \gamma V^{\pi}(f(x,u)) \right]$$

$$\geqslant l(x,\pi') + \gamma Q^{\pi}(x',\pi') = \qquad x' \triangleq f(x,\pi')$$

$$= l(x,\pi') + \gamma l(x',\pi') + \gamma^2 V^{\pi}(x'') \leftarrow \{\text{Cost following } \pi' \text{ for 2 steps and then } \pi$$

$$\geqslant l(x,\pi') + \gamma l(x',\pi') + \gamma^2 Q^{\pi}(x'',\pi') \leftarrow \{\pi' \text{ for 3 steps}$$

$$\geqslant \sum_{i=0}^{\infty} \gamma^i l(x^{(i)},\pi') = V^{\pi'}(x) \implies V^{\pi} \geqslant V^{\pi'} \iff \pi' \leq \pi$$

MODIFIED POLICY ITERATION

- Evaluating exactly a policy can take many iterations
- What if we compute an approximation of V^{π_k}
  - Use m_k policy evaluation iterations to compute V^{π_k}
- Under some mild assumptions it converges to V^*
- For $m_k = \infty$ we recover Policy Iteration
- For $m_k = 1$ we get Value Iteration
  - every time you update V you use a policy that's greedy w.r.t. V

CONTROL: VALUE ITERATION

Algorithm:

- Start with arbitrary value function V_0
- For $k = 0 \dots \infty$
  - For any $x \in \mathcal{X}$
    - $V_{k+1}(x) = \min_{u \in \mathcal{U}} l(x,u) + \gamma V_k(f(x,u))$

$\Big\}$ $V_{k+1} = T V_k$

- V_k converges to V^*: $\lim_{k \to \infty} V_k = V^*$
- Optimal policy π^* is computed at the end from V^*

$$\pi^* = \arg\min_u \; l + \gamma V^*$$

CONVERGENCE OF VALUE ITERATION

Three-step prove:

① Prove T is contracting

② Prove $V_{k+1} = T V_k$ converges to fixed point $V = TV$

③ Prove T has unique fixed point V

① Prove T is contracting, i.e. $\|TZ - TV\|_\infty \leq \gamma \|Z - V\|_\infty$

Remember definition of T: $TV = \min_u C^u + \gamma P^u V$

$$\|TZ - TV\|_\infty = \|\min_u [C^u + \gamma P^u Z] - \min_w [C^w + \gamma P^w V]\|_\infty =$$

Exploit fact that: $\min_y f(y) = -\max_y [-f(y)]$

$$= \|-\max_u(-Q^z) + \max_w(-Q^v)\|_\infty =$$

$$= \|\max_w(-Q^v) - \max_u(-Q^z)\|_\infty =$$

Exploit that: $|\max_y f(y) - \max_w g(w)| \leq \max_y |f(y) - g(y)|$

$$\leq \|\max_u [-C^u - \gamma P^u V + C^u + \gamma P^u Z]\|_\infty =$$

$$= \gamma \|\max_u P^u (Z - V)\|_\infty \leq \gamma \|\max_u P^u |Z - V|\|_\infty =$$

The infinity norm takes the max over the states, so:

$$= \gamma \max_{x,u} P^u_x |Z - V|$$

Exploit the fact that: $|Z(x) - V(x)| \leq \|Z - V\|_\infty \quad \forall x \in X$

$$\leq \gamma \max_{x,u} P^u_x \mathbf{1} \|Z - V\|_\infty$$

$\mathbf{1}$ = vector with all 1's

Since each row of P^u sums to 1:

$$= \gamma \|Z - V\|_\infty$$

② Assume $V_{n+1} = T V_n$, and V is fixed point of T: $TV = V$

$$\|V_n - V\| = \|TV_{n-1} - TV\|$$

$$\leq \gamma \|V_{n-1} - V\| = \gamma \|TV_{n-2} - TV\|$$

$$\leq \gamma^2 \|V_{n-2} - V\| = \dots$$

$$\leq \gamma^n \|V_0 - V\| \Rightarrow \lim_{n\to\infty} \|V_n - V\| = 0$$

So V_n converges to a fixed point of T at a geometric rate.

Finally let's show that T has a unique fixed point.

Let's assume V and Z are both fixed points of T:

$$\|V - Z\| = \|TV - TZ\| \leq \gamma \|V - Z\| \Rightarrow (1-\gamma)\|V - Z\| \leq 0$$

$$\Rightarrow \|V - Z\| = 0 \Rightarrow V = Z$$

MODEL-FREE PREDICTION

Problem: Find Value function of unknown MDP by acting and observing

- Monte Carlo (MC) methods can solve this problem
- Simple idea: Value = mean return
- Caveat: can only be applied to episodic MDP
  - All episodes must end

MONTE CARLO POLICY EVALUATION

Goal: learn V_π from episodes of experience under policy π

$$x_1, u_1, l_1, \ldots, x_n \sim \pi$$

Cost-to-go is total discounted cost:

$$J_t = l_t + \gamma l_{t+1} + \ldots + \gamma^{T-1} l_{T-1}$$

Value function is expected cost-to-go under policy π:

$$V^\pi(x) = \mathbb{E}_\pi[J_t \mid x_t = x]$$

← Implicitly assume some stochasticity (dynamics, policy, cost)

MC policy evaluation estimates V^π as the average cost-to-go.

FIRST-VISIT MC POLICY EVALUATION

The first time that a state x is visited in an episode:

- Increment counter $N(x) := N(x) + 1$
- Increment total cost $C(x) := C(x) + J(x)$
- estimate value as $V(x) = C(x) / N(x)$

By law of large numbers $V(x) \to V^{\pi}(x)$ as $N(x) \to \infty$

Another version exists where counter and total cost are incremented every time a state is visited (every visit).

INCREMENTAL MC UPDATES

IDEA: Compute mean incrementally:

$$V_N = \frac{1}{N}\sum_{j=1}^{N} J_j = \frac{J_N + \sum_{j=1}^{N-1} J_j}{N} = \frac{J_N + (N-1)\,V_{N-1}}{N} = V_{N-1} + \frac{1}{N}\left(J_N - V_{N-1}\right)$$

In non-stationary problems it can be useful to track a running mean, i.e. Forget old episodes

$$V(x) := V(x) + \alpha\left(J - V(x)\right)$$

TEMPORAL DIFFERENCE LEARNING - TD(0)

- Alternative to MC
- Estimate cost-to-go by 1-step lookahead (TD(0)):

$$V(x_t) := V(x_t) + \alpha_t\big(\underbrace{\underbrace{\ell_t + \gamma V(x_{t+1})}_{\text{TD target} = \text{estimated return}} - V(x_t)}_{\text{TD error}}\big)$$

- Compare to MC incremental update:

$$V(x_t) := V(x_t) + \alpha_t\big(J_t - V(x_t)\big)$$

- Policy evaluation is TD(0) with $\alpha_t = 1$ and sampling all states uniformly

TD(0) PROPERTIES

- TD(0) is stochastic approximation algorithm
- Since Bellman operator has unique fixed point => If TD(0) converges (and all states are sampled infinitely often) then it must converge to V^{π}.
- Convergence is guaranteed if step-size sequence satisfies Robbins-Monro (RM) conditions

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

[Ex. $\alpha_t = \frac{\bar{\alpha}}{t}$]

- In practice people often use constant step sizes

MC vs TD

- TD can learn after every step
- MC must wait until end of episode to know cost

- TD works in non-terminating environments
- MC only works for episodic (terminating) environments

- TD target is biased estimate of $V_\pi(x_t)$
- Cost-to-go J_t is unbiased estimate of $V_\pi(x_t)$

- TD target is lower variance than J_t ← monte carlo target
  - J_t depends on many random transitions
  - TD target depends on one random transition

MC vs TD

- MC has high variance, zero bias
  - good convergence properties
  - (even with function approximation)
  - not very sensitive to initialization
  - simple to understand and use
- TD has low variance, some bias
  - usually more efficient than MC
  - TD(0) converges to $V_\pi(x_T)$
  - (but not always with function approximation)
  - more sensitive to initial value

AB EXAMPLE

Two states (A, B), no discounting, 8 episodes

A, 0, B, 0

B, 1

B, 1

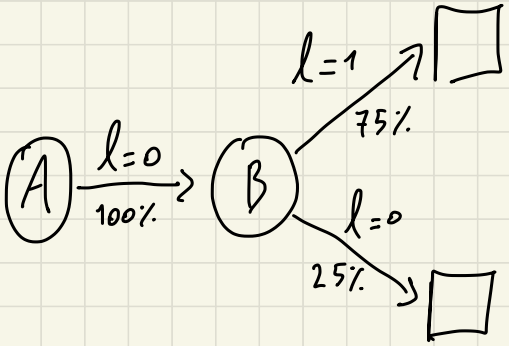
B, 1

B, 1

B, 1

B, 1

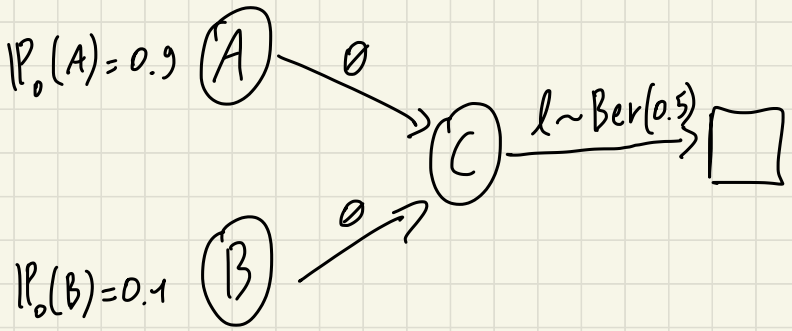
B, 0



$$\begin{cases} V(B) = 0.75 \\ V(A) = \nearrow 0 \text{ for MC} \\ \qquad\quad \searrow 0.75 \text{ for TD}(0) \end{cases}$$

What's V(A), V(B)?

ABC EXAMPLE



- After k visits of B, C has been visited $\approx 10k$ times
- $Var[\hat{V}(C)] \approx 1/(10k)$

- With TD(0) the variance of the TD Target for V(B) decreases with the # of episodes

$$\hat{V}(B) := \hat{V}(B) + \alpha\left(0 + \hat{V}(C) - \hat{V}(B)\right)$$

- With MC the variance of the Target is always 0.25
  - MC is slower to converge in this example

- Assume now the cost after C is deterministically 1
- MC becomes faster because it immediatly gets the true target value
- TD(0) has to wait for $\hat{V}(C)$ to converge to 1 before the target of V(B) gets close to 1
  - Things would be even worse if we had a long chain of states with non-zero cost only at the end

BOOTSTRAPPING & SAMPLING

Bootstrapping: updates involve an estimate

Sampling: updates sample an expectation

Boot str. / Sampl.	NO	YES
NO	Exhaustive search	DP
YES	MC	TD

n-STEP RETURN

$n = 1$ (TD) $J_t^{(1)} = l_t + \gamma V(x_{t+1})$

$n = 2$ $J_t^{(2)} = l_t + \gamma l_{t+1} + \gamma^2 V(x_{t+2})$

$\vdots$ $\vdots$

n $J_t^{(n)} = l_t + \gamma l_{t+1} + \dots + \gamma^{n-1} l_{t+n-1} + \gamma^n V(x_{t+n})$

$\vdots$ $\vdots$

$n = \infty$ (MC) $J_t^{(\infty)} = l_t + \gamma l_{t+1} + \dots + \gamma^{T-1} l_{t+T-1}$

- n-step TD learning:

$$V(x_t) := V(x_t) + \alpha_t (J_t^{(n)} - V(x_t))$$

MODEL FREE CONTROL
CONTROL LEARNING METHODS

		LEARNED QUANTITIES	
		$V(x)$ / $Q(x,u)$	$V(x)$ / $Q(x,u)$ + $\pi(x)$
MDP	Known	Value Iteration	Policy Iteration
	Unknown	Direct methods	Actor-Critic methods

- Why learn Q instead of V?

$$\pi(x) = \arg\min_u \; \ell(x,u) + \gamma V(x') = \arg\min_u \; Q(x,u)$$

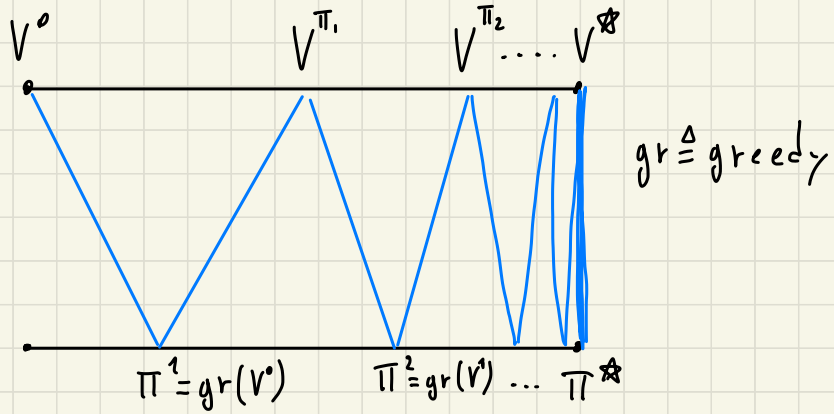
Action-value Function

- Cannot compute x' if MDP is unknown!

MODEL FREE CONTROL

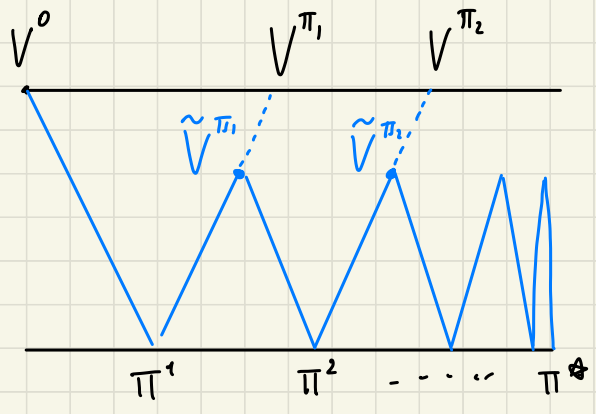
ACTOR-CRITIC METHODS: GENERALIZED POLICY ITER.

- Extension of Policy Iteration to unknown MDP
- PI alternates "policy evaluation" and "policy improvement"
- Unknown MDP => use MC or TD to evaluate policy
- Exact policy evaluation can take infinitely many samples
- Solution: improve policy based on approximate value function



$gr \triangleq$ greedy

PI: exact policy evaluation



GPI: approximate policy evaluation

- The value function is called "critic" because it evaluates the policy, which is called "actor", in order to improve it
- The policy used to generate transition is typically not the same that is evaluated and improved
  - The "behavior policy" mixes small amount of exploration into "target policy"
- GPI can generate policies worse than previous ones

SARSA: CRITIC

- Use TD(0) to learn Q from on-policy samples
- After each transition (x, u, ℓ, x', u'), i.e. (s, a, r, s', a'), update Q as follows:

$$\delta(Q) := \ell(x,u) + \gamma Q(x',u') - Q(x,u)$$

$$Q_{k+1}(x,u) := Q_k(x,u) + \alpha_k \delta(Q)$$

- If π was fixed $\Rightarrow$ SARSA = TD(0)

- As TD, SARSA can diverge in off-policy situations

- How to improve policy?

HOW TO CHOOSE CONTROL?

- What about being greedy w.r.t. $Q_n(\cdot)$? $u(x) = \arg\min_w Q_n(x, w)$

EXAMPLE

- You can choose one of two doors in front of you
  - Left door $\Rightarrow$ $\ell = 0$ $\Rightarrow$ $V(\text{left}) = 0$
  - Right door $\Rightarrow$ $\ell = -1$ $\Rightarrow$ $V(\text{right}) = -1$
  - Right door $\Rightarrow$ $\ell = -3$ $\Rightarrow$ $V(\text{right}) = -2$

Are you sure the right door is the best one?

SARSA: ACTOR

- Simplest idea: take policy that is **greedy** w.r.t. Q
  - Greedy policy can be computed on the fly => faster!
- **GREEDY** STRATEGY:
  - Always choose control with best estimated cost
  - Can fail to find best action => large loss over time
  - Good learner must choose suboptimal control to **explore**
- **ε-GREEDY** STRATEGY:
  - Choose **random** control with **probability ε**
  - Choose **greedy** control with **probability $1-\varepsilon$**
  - Simple way to ensure exploration
  - ε can change over time (typically decreasing to 0)

SARSA: CONVERGENCE

- Use ε-greedy as behavior policy to ensure exploration
- SARSA converges to $Q^*(x,u)$ if:
  1. Greedy in the Limit with Infinite Exploration (GLIE) policies:
     - All state-action pair are visited infinitely many times
     - Policy converges on greedy policy (e.g.: $\varepsilon_k = \frac{1}{k}$)
  2. Robbins-Monro sequence of step sizes:

$$\sum_{k=1}^{\infty} \alpha_k = \infty \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

What if we want to learn off-policy?

ON-POLICY vs OFF-POLICY LEARNING

- **On-policy** (e.g. SARSA)
  - learn about policy π from experience sampled from π
  - behavior policy $\approx$ target policy (e.g. ε-greedy, with ε small)
  - limited exploration => low sample efficiency

- **Off-policy**
  - learn about policy π from experience sampled from μ
  - behavior policy μ can be arbitrarily different from target policy π
  - higher sample efficiency

DIRECT METHODS: Q LEARNING

- Iteratively update an estimate $Q_k(x,u)$ of $Q^*(x,u)$
- After observing transition (x, u, ℓ, x') update with:

$$\delta(Q) := \ell + \gamma \min_{w \in U} Q(x', w) - Q(x, u)$$

$$Q_{k+1}(x, u) := Q_k(x, u) + \alpha_k \delta(Q_k)$$

- Same as SARSA, but uses $\min_w Q(x', w)$ instead of $Q(x', u')$
- Error $\delta(Q(x,u))$ is independent of behavior policy, which is the one that generates u' => OFF-POLICY
- Target policy is: $\pi(x) = \arg\min_u Q(x, u)$

Q-LEARNING: CONVERGENCE

Need exploration!

- At stochastic equilibrium, $\forall (x,u)$ visited infinitely often:

$$\mathbb{E}[\delta(Q)] = 0 = TQ - Q \Rightarrow Q = Q^* \Leftarrow \text{Unique fixed point of } T$$

- Q_k converges when appropriate learning rates are used (Robbins-Monro conditions) and sufficient exploration is ensured

SUMMARY

DP	TD
Policy Evaluation	**TD Learning**
$V(x) = \ell + \gamma V(x')$	$V(x) \overset{\alpha}{\leftarrow} \ell + \gamma V(x')$
Policy Iteration	**Sarsa**
$V(x) = \ell + \gamma V(x')$	$Q(x,u) \overset{\alpha}{\leftarrow} \ell + \gamma Q(x',u')$
$\pi(x) = \arg\min_u \ell + \gamma V(x'(u))$	$\pi(x) = \arg\min_u Q(x,u)$
Value Iteration	**Q-Learning**
$V(x) = \min_u \ell + \gamma V(x'(u))$	$Q(x,u) \overset{\alpha}{\leftarrow} \ell + \gamma \min_{u'} Q(x',u')$

VALUE FUNCTION APPROXIMATION

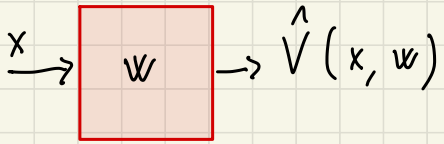
- Discrete-space algorithms don't scale to large systems:
  - Backgammon: 10^{20} states
  - Go: 10^{170} states
  - Robots: continuous state space
- Too many states/control to store and learn
- IDEA: Use function approximation to represent $V(x)$ / $Q(x,u)$

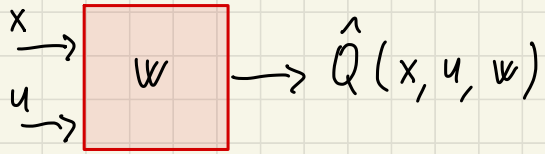
$$\hat{V}(x, w) \approx V(x)$$

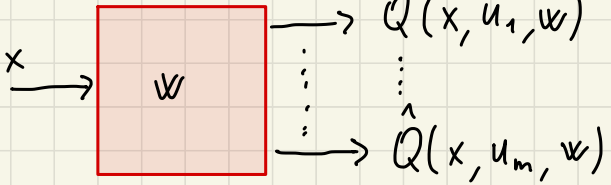
$$\hat{Q}(x, u, w) \approx Q(x, u)$$

- w = parameters defining the function

TYPES OF VALUE FUNCTION APPROXIMATION





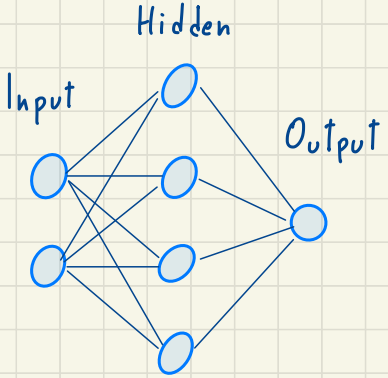


- Neural networks are often used as function approximation
- Any differentiable approximator can be used:
  - linear combination of features
  - Fourier basis
  - Polynomials

Arbitrary user-defined functions of x

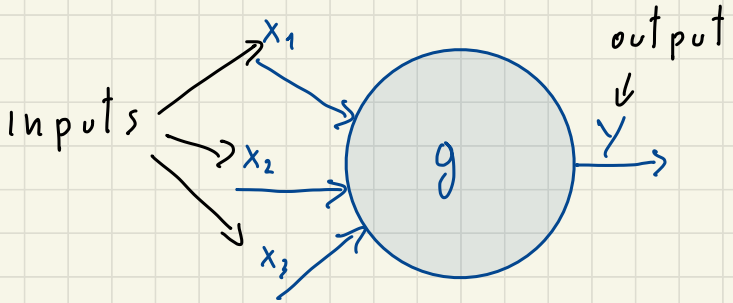
- Problems:
  - non-stationary target V
  - non i.i.d. data

NEURAL NETWORKS



- Parametric function structured in "layers":
  - 1 input layer (size = # of inputs)
  - ≥ 1 hidden layers
  - 1 output layer (size = # of outputs)
- Each layer is composed of "neurons"
- In fully-connected networks each neurons recieves as input the outputs of all neurons of the previous layer
- A neuron outputs a nonlinear function of a linear combination of its inputs

NEURONS



$$y = g\left(\sum_i w_i x_i + w_0\right)$$

- $g(\cdot)$ is the nonlinear "activation" function, e.g.:
  - ReLU, Rectified Linear Unit: $g(p) = \max(0, p)$
  - Hyperbolic tangent: $g(p) = \tanh(p) = \frac{e^p - e^{-p}}{e^p + e^{-p}}$
- w is the vector of weights associated to the inputs x
- w_0 is called the "bias"

TRAINING A NEURAL NETWORK

- Training = Finding the values of all weights w
- Typically formulated as unconstrained optimization:

$$\underset{w}{\text{minimize}} \sum_i \| \phi(x_i; w) - y_i \|^2 \triangleq J(w)$$

LOSS FUNCTION

- Given a list of input-output pairs (x_i, y_i), find w such that the network's output $\phi(x; w)$ is close to y.
- Since size(w) can be very large, typically solved with some variants of gradient descent:

$$w := w - \alpha \nabla J(w)$$

α = step size

STOCHASTIC GRADIENT DESCENT

- If the number of input-output pairs is large, computing ∇J can be computationally expensive

$$\nabla_w J = \sum_{i \in \mathcal{D}} 2\left(\phi(x_i; w) - y_i\right) \nabla_w \phi(x_i; w)$$

- IDEA: Compute ∇J using only a subset of the data
- In the limit the subset (mini batch) can be just one sample
- Gradient is approximated (i.e. noisy) so:
  - more iterations needed to converge
  - each iteration is much cheaper to compute

INCREMENTAL PREDICTION ALGORITHMS

- How can we get target $y = V(x)$?
- For MC, target is cost-to-go $J_t(x)$:

$$\Delta w = \alpha \left(J_t - \phi(x_t, w) \right) \nabla_w \phi(x_t, w)$$

- MC converges to local optimum even with nonlinear approximators
- For TD(0), target is TD target:

$$\Delta w = \alpha \left(l_t + \gamma \phi(x_{t+1}, w) - \phi(x_t, w) \right) \nabla_w \phi(x_t, w)$$

- TD converges (close) to global optimum with linear approxim.

INCREMENTAL CONTROL WITH FUNCTION APPROXIM.

IDEA: Generalized Policy Iteration with approximate Action-Value function:

$$\min_{w} \mathbb{E}_{\pi}\left[\left(\phi(x, u, w) - Q_{\pi}(x, u)\right)^2\right]$$

- Use SGD to find local minimum
  - For MC, target is cost-to-go $J_t(x_t, u_t)$
  - For TD(0), target is $\ell_t(x_t, u_t) + \gamma \phi(x_{t+1}, u_{t+1}, w)$
- Use ε-greedy policy to ensure exploration
  - Greedy control is easily computed for discrete control space

CONVERGENCE OF CONTROL ALGORITHMS

	VALUE ENCODING		
	Table Lookup	Linear	Nonlinear
MC control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-Learning	✓	✗	✗

(✓) = chatter around optimum

Linear $\Rightarrow$ $\phi(x, w) = h(x)^T w$ $\quad$ $h(x)$ = nonlinear function

LIMITATIONS OF INCREMENTAL CONTROL

- Each transition (x, u, ℓ, x') is used to perform a single gradient step on w and then thrown away

  => POOR DATA EFFICIENCY

- Consecutive transitions are strongly correlated because the state evolves continuously

  => DATA IS NOT INDEPENDENTLY & IDENTICALLY DISTRIBUTED

Can we overcome these issues?

BATCH LEARNING

- IDEA: Instead of learning incrementally, learn after having collected a batch of training data

- Least-squares prediction:

$$\min_{w} \sum_{t=1}^{T} \left(V_t - \phi(x_t, w) \right)^2$$

- How to achieve it? Experience replay:
  - store data $\langle x_t, V_t \rangle$ in replay buffer
  - sample minibatch from buffer
  - apply SGD to update w
  - converge to LS solution

DEEP Q-NETWORK (2015, Mnih et al.)

- Extension of Q-Learning to continuous state space
- Discrete control space
- Use neural network to represent $Q(x, u, w)$
- ε-greedy policy with ε decreasing over time
- Use experience replay:
  - Store transitions (x_t, u_t, l_t, x_{t+1}) in replay buffer
  - Every few steps (e.g. 4) sampled randomly from replay buffer a mini-batch of transitions (size 32)
  - Use stochastic gradient descent to optimize w

DQN uses fixed Q targets to stabilize training:

- compute Q targets using old fixed parameters w^-
- optimize MSE but Q-network and Q-learning targets:

$$\underset{w}{\text{minimize}}\ \mathbb{E}\left[\ell + \gamma \min_{u'} \phi(x', u', w^-) - \phi(x, u, w)\right]$$

- every C steps update target weights:

$$w^- := w$$

- Store w resulting in best performance as performance may get worse during training

CONCLUSIONS

- Introduce function approximation to scale RL algorithms
- Use stochastic gradient descent for training neural networks
- Incremental alg. update w at every step with current transition
- Batch alg. introduce replay buffer to improve sample efficiency and break data dependency
- Action space is still assumed to be discrete

Policy Gradient Methods: Direct Policy Optimization

Andrea Del Prete

December 5, 2025

Optimization and Learning for Robot Control—University of Trento

Table of contents

From Value-Based to Policy-Based Methods

Context: From Value-Based to Policy-Based Methods

Review: So far, we have studied Value-Based methods (Q-Learning, SARSA, DQN).

- These methods learn action values ($Q(x, u)$) and select actions based on them (e.g., ϵ-greedy). Their policies would not exist without the action-value estimates.

Policy Gradient Methods (PGM):

- Learn a **parameterized policy** $\pi(u|x, \theta)$ directly, which selects actions without consulting a value function.
- A value function may still be used to learn the policy parameter, but it is not required for action selection.
- PGM seek to **minimize cost** $J(\theta)$ using gradient descent: $\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$.

Policy Parameterization for Continuous Actions

Policy-based methods naturally handle **continuous action spaces** by learning the statistics of the probability distribution, rather than individual action probabilities.

Gaussian Policy Parameterization: The policy $\pi(u|x,\theta)$ can be defined as a normal (Gaussian) probability density function over a real-valued action u:

$$\pi(u|x,\theta) \propto \exp\left(-\frac{(u-\mu(x,\theta))^2}{2\sigma(x,\theta)^2}\right)$$

Where the policy parameter vector can be divided into two parts, $\theta = [\theta_\mu, \theta_\sigma]$.

- **Mean Approximation**: $\mu(x,\theta_\mu)$.
- **Standard Deviation Approximation**: $\sigma(x,\theta_\sigma)$.

Why Optimize the Policy (π)? I: Continuous Action Spaces

For robotics systems, actions are usually continuous (e.g., joint torques, joint velocities).

- Value-based methods require efficiently solving $\text{argmin}_u Q^\pi(x, u)$. This is a **challenge for continuous or high-dimensional action spaces**.
- Policy-based methods offer practical ways of dealing with large/continuous action spaces.

Key Policy Advantages:

- naturally handle continuous action spaces.
- optimize what we care about (the performance measure $J(\theta)$).

Why Optimize the Policy (π)? II: Stochasticity and Simplicity

1. **Intrinsically Stochastic Policies**:
   - PGM use a stochastic policy class $\pi(u|x, \theta)$ that is differentiable with respect to θ.
   - The action probabilities change smoothly as a function of the learned parameter θ.
   - PGM enable the selection of actions with arbitrary probabilities and can find stochastic optimal policies (e.g., when bluffing in Poker).
   - PGM can approach deterministic policies asymptotically, if optimal.
2. **Simplicity of the Policy Function**:
   - The policy function π may be a **simpler function to approximate** than the value function Q or V.

Policy Gradient Theorem

Objective: Minimizing Global Cost $J(\theta)$

Policy Search Formalization:

- A trajectory τ is a sequence of states and actions (e.g., a robot's history).
- $C(\tau)$ is the corresponding cost-to-go.
- π_θ is the policy parameterized by θ.

Objective Function $J(\theta)$: We aim to minimize the expected cost-to-go $J(\theta)$, defined as the sum of costs over all possible trajectories, weighted by their probability of occurrence under policy π_θ :

$$\theta^* = \text{argmin}_\theta \, J(\theta) = \text{argmin}_\theta \, E_{\tau \sim \pi_\theta}[C(\tau)]$$

$$J(\theta) = \sum_\tau P(\tau|\theta) C(\tau)$$

Solution: Use Gradient Descent to find θ^*:

$$\theta' = \theta - \alpha \nabla_\theta J(\theta)$$

Step 1: Calculating the Gradient (Using the Gradient of Sum)

We start by taking the gradient of $J(\theta)$ with respect to θ:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau P(\tau|\theta) C(\tau)$$

We can swap the gradient and the summation (the gradient of a sum is the sum of gradients):

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta P(\tau|\theta) C(\tau)$$

Challenge: The term $\nabla_\theta P(\tau|\theta)$ is hard to compute directly, as it depends on the unknown environment dynamics and the policy π_θ.

Step 1: Gradient Calculation

To transform the sum into a sampleable expectation, we use the Likelihood Ratio trick (or REINFORCE trick).

We multiply by 1, in the form $\frac{P(\tau|\theta)}{P(\tau|\theta)}$:

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau|\theta) \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} C(\tau)$$

Recall the logarithm identity: $\nabla_\theta \log x = \frac{\nabla_\theta x}{x}$.

Substituting this identity, we get the gradient as an Expectation:

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) C(\tau)$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta}[\nabla_\theta \log P(\tau|\theta) C(\tau)]$$

Intuition: This gradient tries to increase the probability of paths (τ) with negative costs $C(\tau)$ and decrease the probability of paths with positive costs.

Step 2: Decomposing the Trajectory Probability $P(\tau|\theta)$

A trajectory τ is defined by the sequence $x_1, u_1, l_1, x_2, u_2, l_2, \ldots$ over horizon H .

Under the Markov assumption, the probability of a trajectory $P(\tau|\theta)$ is the product of the transition probabilities $p(x_{t+1}|x_t, u_t)$ and the action probabilities defined by the policy $\pi_\theta(u_t|x_t)$:

$$P(\tau|\theta) = \prod_{t=1}^{H} p(x_{t+1}|x_t, u_t) \cdot \pi_\theta(u_t|x_t)$$

We now apply the logarithm and the gradient to this expression :

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \log \left[\prod_{t=1}^{H} p(x_{t+1}|x_t, u_t) \cdot \pi_\theta(u_t|x_t) \right]$$

Using the property that the log of a product is the sum of logs :

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \left[\sum_{t=1}^{H} \log p(x_{t+1}|x_t, u_t) + \sum_{t=1}^{H} \log \pi_\theta(u_t|x_t) \right]$$

Step 2: Eliminating Unknown Dynamics

The crucial step relies on the fact that the gradient (∇_θ) operates only on the policy parameters θ .

- The **transition probability** $p(x_{t+1}|x_t, u_t)$ depends on the environment dynamics, but it **does not** depend on the policy parameters θ.
- Therefore, the gradient of the first term is zero: $\nabla_\theta \log p(x_{t+1}|x_t, u_t) = 0$.

This simplifies the expression significantly :

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \sum_{t=1}^{H} \log \pi_\theta(u_t|x_t)$$

$$\nabla_\theta \log P(\tau|\theta) = \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(u_t|x_t)$$

The Key Insight: The environment dynamics model is **not required** to compute the gradient!

Final Policy Gradient (Plain Policy Gradient)

Substituting the final expression for $\nabla_\theta \log P(\tau|\theta)$ back into the expectation equation, we obtain the Policy Gradient Theorem (in its simple likelihood ratio form):

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta}\left[\left(\sum_{t=1}^{H}\nabla_\theta \log \pi_\theta(u_t|x_t)\right) C(\tau)\right]$$

Monte Carlo Approximation : The expectation can be approximated by sampling m trajectories $(\tau^{(i)})$ from the current policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H}\nabla_\theta \log \pi_\theta(u_t^{(i)}|x_t^{(i)})C(\tau^{(i)})$$

This result forms the basis for the REINFORCE algorithm (Monte Carlo Policy Gradient).

The REINFORCE algorithm

REINFORCE: Derivation of the Update Rule

The final expression for the gradient expectation is:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_\theta \log \pi(u_t^{(i)} | x_t^{(i)}, \theta) C(\tau^{(i)})$$

The REINFORCE Update Rule:

$$\theta_{t+1} \leftarrow \theta_t - \alpha C_t \nabla \log \pi(U_t | X_t, \theta)$$

Intuition of the Update:

- The update moves θ in the direction that **increases the log-probability** of the action U_t taken, proportional to the negative cost-to-go C_t.
- If C_t is negative, the probability of U_t is increased. If C_t is positive, the probability is decreased.
- This gradient tries to increase the probability of paths (τ) with negative cost-to-go and decrease the probability of paths with positive cost-to-go.

REINFORCE: Key Characteristics

REINFORCE is a Monte Carlo (MC) Algorithm:

- It uses the **complete cost-to-go** C_t (the total discounted cost from time t until the end of the episode).
- Updates must be made in retrospect **after the episode is completed**.

Theoretical Property:

- REINFORCE is a **stochastic gradient method** whose expected update over an episode is in the opposite direction of the cost-to-go's gradient.
- It has good theoretical convergence properties (converging to a local optimum under standard conditions for decreasing α).

REINFORCE (Monte Carlo Policy Gradient)

```
REINFORCE: Monte-Carlo Policy-Gradient Control (episodic)
Input: A differentiable policy parameterization pi(u|x, theta)
Parameter: step size alpha > 0
Initialize policy parameter theta (e.g., to 0)

For each episode:
  Generate an episode X[0], U[0], L[0], ... L[T-1], following pi
  C <- Compute cost-to-go
  For each step t = 0, 1, ..., T - 1:
    # Policy Update
    theta -= alpha * C * nabla ln pi(Ut | Xt, theta)
```

Reducing the variance

The Problem of High Variance

The Trade-off: REINFORCE is unbiased, but highly inefficient.

- Although the estimator is **unbiased** (its expectation is proportional to the true gradient), as a Monte Carlo method, it uses the complete cost-to-go G_t from a single rollout.
- The cost-to-go C_t is highly stochastic, meaning the sample gradient estimate is **very noisy** (high variance).
- This high variance leads to slow learning.

Goal: Find a way to reduce the variance of the gradient estimate without introducing bias.

Exploiting Causality

The action at time t only affects the costs at times $t' \geq t$:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=1}^{H} \nabla_\theta \log \pi(u_t^{(i)} | x_t^{(i)}, \theta) \right) C(\tau^{(i)})$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=1}^{H} \nabla_\theta \log \pi(u_t^{(i)} | x_t^{(i)}, \theta) \right) \left(\sum_{k=0}^{H} l(x_k^{(i)}, u_k^{(i)}) \right)$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=1}^{H} \nabla_\theta \log \pi(u_t^{(i)} | x_t^{(i)}, \theta) \left(\underbrace{\sum_{k=0}^{t-1} l(x_k^{(i)}, u_k^{(i)})}_{\text{Independent of } u_t^{(i)}} + \sum_{k=t}^{H} l(x_k^{(i)}, u_k^{(i)}) \right) \right)$$

Removing the part of the cost that is independent of the current action can reduce the variance.

Exploiting Causality

The improved policy gradient is:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=1}^{H} \nabla_\theta \log \pi(u_t^{(i)} | x_t^{(i)}, \theta) \left(\sum_{k=t}^{H} l(x_k^{(i)}, u_k^{(i)}) \right) \right)$$

Still unbiased, however, still high variance.

Variance Reduction: Introducing the Baseline

The primary technique to reduce variance without introducing bias is the introduction of an arbitrary **Baseline** $b(x)$.

REINFORCE Update with Baseline:

$$\theta_{t+1} \leftarrow \theta_t - \alpha \left(C_t - b(X_t)\right) \nabla \ln \pi(U_t|X_t, \theta)$$

The term $(C_t - b(X_t))$ is now the "effective cost-to-go" used to scale the gradient step.

Baseline Validity: The Unbiased Property

The generalized PGT remains valid **if and only if** the baseline $b(x)$ does not vary with the action u.

The change in the gradient expectation due to the baseline is zero:

$$E_\pi \left[\nabla \ln P(\tau|\theta) b(x)\right] = \sum_\tau P(\tau|\theta) \nabla \ln P(\tau|\theta) b(x)$$

$$= \sum_\tau \nabla P(\tau|\theta) b(x) = \nabla \left(\sum_\tau P(\tau|\theta) b(x) \right) =$$

$$= b(x) \nabla \left(\sum_\tau P(\tau|\theta) \right) = 0$$

Since the sum of probabilities of all trajectories must equal one ($\sum_\tau P(\tau|\theta) = 1$), its gradient is null ($\nabla(\sum_\tau P(\tau|\theta)) = 0$).

The expected value of the subtracted baseline term is zero. Thus, the gradient estimate remains **unbiased**.

Baseline Choice: The State-Value Function

Natural Choice: The most effective choice for the baseline is an estimate of the **state-value function**, $V(X_t, w)$.

The Advantage Function ($A(x, u)$): When $b(X_t) = V(X_t, w)$, the term scaling the gradient is an estimate of the Advantage function:

$$\hat{A}(X_t, U_t) = C_t - V(X_t, w)$$

Intuition: The policy probability $\pi(U_t|X_t, \theta)$ is increased only if the action U_t yields a cost-to-go C_t that is **better** than the expected cost-to-go $V(X_t)$.

- This relative comparison allows the parameter θ to differentiate between better and worse actions in a state where all returns might be generally high or generally low.
- Empirically, adding the state-value function as a baseline can make REINFORCE learn much faster.

REINFORCE with Baseline (Pseudocode)

```
Input: Policy pi_theta (e.g., Neural Network)
Initialize policy parameters theta, baseline b

Loop forever (for each iteration/batch):
  1-Collect a batch of m trajectories {tau^(i)} by running
     current policy pi_theta
  2-For each time step calculate the total discounted
     cost-to-go C_t and the advantage A_t = C_t-b(X_t)
  3-Re-fit the baseline by minimizing ||b(X_t) - C_t||
  4-Update the policy  using a policy-gradient estimate
     SUM_i=1^m SUM_t=1^H [log(pi(u_t^(i) | x_t^(i))) * A_t^(i)]
```

Actor-Critic Policy-Gradient Methods

From REINFORCE to Actor-Critic Methods

REINFORCE with Baseline still relies on the **full Monte Carlo cost-to-go** (C_t), meaning updates are delayed until the end of the episode.

Actor-Critic Goal: Achieve online, incremental learning and further reduce variance by introducing **bootstrapping**.

- **Actor (π_θ)**: The policy that selects actions.
- **Critic ($V_w(x)$)**: The learned value function that evaluates the policy's actions (assigns credit).

The Trade-off:

- Using bootstrapping (updating the value estimate from estimated values of subsequent states) reduces variance and accelerates learning.
- However, bootstrapping introduces **bias** and an asymptotic dependence on the quality of the function approximation.

One-Step Actor-Critic Methods

We replace the full Monte Carlo cost-to-go C_t with the **one-step cost-to-go** $C_{t:t+1}$ to enable bootstrapping and online updates.

The one-step Advantage estimate is the Temporal Difference (TD) error (δ_t):

$$\delta_t = L_{t+1} + \gamma V(X_{t+1}, w) - V(X_t, w)$$

Policy Update (Actor):

$$\theta_{t+1} \leftarrow \theta_t - \alpha_\theta \delta_t \nabla \ln \pi(U_t | X_t, \theta)$$

Value Function Update (Critic): The critic is typically updated using TD(0):

$$w_{t+1} \leftarrow w_t + \alpha_w \delta_t \nabla V(X_t, w)$$

Benefit: This provides a fully online and incremental algorithm, processing states, actions, and costs as they occur.

One-step Actor-Critic (Pseudocode)

```
One-step Actor-Critic (episodic)

For each episode:
  Initialize X
  I <- 1
  Loop while X is not terminal:
    U <- pi(·|X, theta)
    Take action U, observe X', L
    # Calculate TD Error (Delta)
    Delta <- L + gamma * V_hat(X', w) - V_hat(X, w)
    # Critic Update
    w += alpha_w * Delta * nabla V_hat(X, w)
    # Actor Update
    theta -= alpha_t * I * Delta * nabla ln pi(U|X, theta)
    I <- gamma * I
    X <- X'
```

*Note: I is the eligibility trace factor that handles discounting.

Advanced PG: Sophisticated Advantage Estimation

One-step TD error is low variance but high bias. MC cost-to-go (C_t) is high variance but low bias. Advanced methods combine these:

1. **Generalized Advantage Estimation (GAE)**:
   - GAE uses an **exponentially weighted average** of n-step TD errors, similar to TD(λ).
   - GAE balances the bias-variance trade-off by combining returns of different lengths, providing a robust Advantage estimate.

2. **Asynchronous Advantage Actor Critic (A3C)**: Uses n-step Advantage estimation (bootstrapping).

3. **TRPO and PPO**: Introduce constraints (Trust Regions) to stabilize large gradient steps, which is crucial for complex tasks like robotic locomotion.

Summary of Key Concepts

1. **Policy Optimization**: Directly optimizes performance $J(\theta)$ with respect to policy parameters θ. Compatible with continuous actions.
2. **PGT**: Provides the theoretical foundation, guaranteeing that the gradient can be estimated from samples without needing to model state distribution derivatives.
3. **REINFORCE**: A Monte Carlo (MC) method derived from the PGT. It uses the full cost-to-go C_t and is **unbiased** but suffers from high variance.
4. **Baseline**: Subtracting the value function $V^\pi(x)$ reduces variance without introducing bias (unbiased). This yields the Advantage $C_t - V(X_t)$.
5. **Actor-Critic**: Uses the learned value function (Critic) for **bootstrapping** (e.g., using TD error δ_t) to accelerate learning and enable online updates, at the cost of introducing bias.

Application in Robotics

PG methods are highly relevant to robotics due to their ability to handle continuous, high-dimensional action spaces.

- **Continuous Control**: When controlling robotic manipulators, legged robots, or autonomous vehicles, the actions (torques, velocities) are continuous, where value-based methods struggle.
- **Locomotion**: Advanced PG methods like TRPO and PPO, often combined with GAE, have been essential in learning complex locomotion behaviors for simulated and real-world robots.
- **Historical Success**: Policy gradients were crucial for complex applications like acrobatic helicopter autopilots.

Policy Gradient methods provide a framework for learning sophisticated, continuous control policies that can often be simpler to represent than the corresponding value functions.

Conclusion and Future Outlook

- Policy gradient methods offer a significantly different set of strengths and weaknesses compared to action-value methods (Q-Learning).
- While historically less studied than Q-Learning, they are now a subject of great excitement and ongoing research in Deep RL.

Questions?