Reactive Control

- **Differential Kinematics**: relation ship bt velocity of all the joints & the velocity of the e.e. ← Jacobian

- **Statics**:

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

Power @ ee: $P_e = f^T \dot{p} + m^T \omega$ (moment)

$$= \begin{bmatrix} f^T & m^T \end{bmatrix} \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} = w^T v = w^T J \dot{q}$$

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

Dynamics: non linear equation

$$M(q)\ddot{q} + \underbrace{c(q,\dot{q}) + g(q)}_{h(q,\dot{q})} = \tau + \underbrace{J(q)^T w}_{\text{if wrench applied @ ee}}$$

Joint Motion Control

given a refrence joint trajectory → compute $\tau(t)$

such that $q(t) \simeq q^{ref}(t)$

1. open loop control
2. PID control
3. PD + gravity
4. Inverse Dynamics Control

1. Open loop control:

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

2. PID control: use tracking error, no knowledge of dynamics

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

$K_p, K_d, K_i \Rightarrow$ +ve definate diagonal Matrices

if $\dot{q}^{ref} = 0 \Rightarrow$ guarantee convergence

$$M\ddot{q} + C\dot{q} + g = \tau = K_p e + K_d \dot{e} + K_i \int_0^t e\,ds$$

@ S.S. $\Rightarrow \ddot{q} = \dot{q} = 0 \Rightarrow = K_p e + K_i \int_0^t e\,ds$ derive

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

3. PD + gravity

Use knowledge of Model

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

@ steady state $\Rightarrow g = K_p e + g \Rightarrow K_p \dot{e} = 0$

4a. Inverse Dynamics control same as open loop But added the error tracking

$$\begin{cases} \tau = M\ddot{q}^{des} + h \\ \ddot{q}^{des} \triangleq \ddot{q}^{ref} + K_P(q^{ref} - q) + K_d(\dot{q}^{ref} - \dot{q}) \end{cases}$$

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

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

second order linear

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

- need M, Sh $\Leftarrow$ complete knowledge of model
- Stable if $K_p, K_d > 0$ & diagonal

4b. Variant of Inverse Dynamics

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

feedback terms are not pre-multiplied by mass

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

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

$$\dot{y} = \begin{bmatrix} 0 & I \\ -M^{-1}K_p & -M^{-1}K_d \end{bmatrix}$$

Cartisan-Space Motion Control

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

1. compute $q^{ref}(t)$ s.t. $x^{ref}(t) = FG(q^{ref}(t))$ Forward Geometry

2. Operational Space dynamics ⇒ Inverse Dynamics
3. Inverse Kinematics + Inverse Dynamics

1. Compute $q^{ref}(t)$ s.t. $x^{ref}(t) = FG(q^{ref}(t))$

→ then apply Joint motion control

* restricts to only one solution
* Don't know x_{ref} in advance
* going from $q_{ref} \rightarrow x_{ref}$ requires solving NL equations

2. Operational Space Dynamics [OS inverse Dynamics]

- same as inverse Dynamics Control!

* Take Dynamics of JointSpace & project it onto the ee. space using matrix multiplication
* manipulate the equation to get the relationship bt ee acceleration & joint torques!

$$JM^{-1}[M\ddot{q} + h = \tau]$$

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

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

$$\Lambda\ddot{x} + \Lambda(JM^{-1}h - \dot{J}\dot{q}) = \Lambda JM^{-1}J^{T}f$$

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

$$x = FG(q)$$
$$\dot{x} = J\dot{q}$$
$$\ddot{x} = J\ddot{q} + \dot{J}\dot{q}$$
$$J\ddot{q} = \ddot{x} - \dot{J}\dot{q}$$
$$\Lambda = (JM^{-1}J^{T})^{-1}$$

$$\tau = J^{T}f$$

$$f = \Lambda \ddot{x}^{des} + \mu$$

$$\ddot{x}^{des} = \ddot{x}^{ref} + K_p(x^{ref} - x) + K_d(\dot{x}^{ref} - \dot{x})$$

$$\tau = J^T \Lambda \ddot{x}^{des} + J^T \Lambda (JM^{-1}h - \dot{J}\dot{q})$$

Simplified version

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

$$JM^{-1}M\ddot{q} = JM^{-1}J^T \Lambda \ddot{x}^{des}$$

$$\ddot{x} - \dot{J}\dot{q} = \ddot{x}^{des}$$ very small

3. Inverse Kinematics + Inverse Dynamics

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

$J^+ = J(JJ^T)^{-1}$ Moore-Penrose Pseudo inverse

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

$$\ddot{q} = J^+(\ddot{x}^d - \dot{J}\dot{q}) \rightarrow J\ddot{q} = JJ^+(\ddot{x}^d - \dot{J}\dot{q})$$

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

[Postural Task]

Redundancy $\dim(x) < \dim(q)$

→ need to extend your control loop ⇒ adding an extra objective. Stabilize the motion of the joint system.

$$JM^{-1}[M\ddot{q} + h = \tau] \Rightarrow J\ddot{q} + JM^{-1} = JM^{-1}\tau$$

$$\ddot{x} - \dot{J}\dot{q} + JM^{-1} = JM^{-1}\tau \rightarrow \Lambda\ddot{x} + \mu = \underbrace{\Lambda JM^{-1}}_{J^{T\#}}\tau$$

$\Lambda JM^{-1} \Rightarrow J^{T\#}$ Transpose pseudo inverse

* need a τ that results in $\emptyset$ when $J^{T\#}$ is multiplied

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

$$\tau_0 = (1 - JJ^{T\#})\tau_1$$

verify: $J^{T\#}\tau_0 = (J^{T\#} - J^{T\#}JJ^{T\#})\tau_1 = \emptyset$

$$\tau_1 = M(k_p(q^{ref} - q) - k_d(\dot{q}^{ref} - \dot{q})) + h$$

Joint space Inverse Dynamics

$$\tau = J^T\Lambda\ddot{x}^d + J^T\mu + (1 - J^TJ^{T\#})(M\ddot{q}^d + h)$$

Interaction Control — Regulate contact forces

1. Direct force control

* Desired force → controller to apply that force

2a. Indirect / Impedance Control

* regulate force by controlling relationship bt motion & force.

Desired Dynamics : mass-spring-damper system

$$M\ddot{x} + B\dot{x} + Kx = f \quad (1)$$ ← in cartisan space

M: inertia, B: damper, K: stiffness

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

Real Dynamics:

$$M\ddot{q} + h = \tau + J^T f$$ ← Joint space * JM^{-1}

$$J\ddot{q} + JM^{-1}h = JM^{-1}\tau + JM^{-1}J^T f$$

* $\Lambda = (JM^{-1}J^T)^{-1}$

$$\Lambda\ddot{x} + \Lambda(JM^{-1}h - \dot{J}\dot{q}) = \Lambda JM^{-1}\tau + f$$

$$\Lambda\ddot{x} + \mu = J^{T\#}\tau + f \quad (2)$$ ← Cartisan space

from eq 1: $\ddot{x} = M^{-1}(f - B\dot{x} - Kx)$ substitute in eq 2

$$\Lambda M^{-1}(f - B\dot{x} - Kx) + \mu = J^{T\#}\tau + f$$

$$\Lambda M^{-1}(f - B\dot{x} - Kx) + \mu = J^{T\#}J^T f_m + f$$

$\tau = J^T f_m$

$$\Lambda M^{-1} f + \Lambda M^{-1}(-B\dot{x} - Kx) + \mu = f_m + f$$

$$\Lambda M^{-1}(-B\dot{x} - Kx) + \mu = f_m + (1 - \Lambda M^{-1}) f$$

problem: need to measure

$$f_m = \Lambda \cancel{M^{-1}}^{I}(-B\dot{x} - Kx) + \mu - (1 - \Lambda \cancel{M^{-1}}^{I}) f$$

* If we choose the desired inertia to be close to the real one then $M_d = \Lambda$

$$\therefore f_m = -B\dot{x} - Kx + \mu$$

$$\tau = J^T(-B\dot{x} - Kx + \mu)$$

2b. Simplified Impedance Control

Replace μ w/ h

$$\tau = h - J^T(B\dot{x} + Kx)$$

analytical ↑

numerical ↓

QP-Based Reactive Control

* Numerical optimization problem that take into account the limits of the system

$M(q)\ddot{q} + h(q, \dot{q}) = \tau$ given $q^r(t) \rightarrow$ find τ st $q(t) \simeq q^r(t)$

$$\tau_d = M\ddot{q}^d + h \Rightarrow \ddot{q}^d = \ddot{q}^r + K_p e + K_d \dot{e}$$

→ optimization problem:

$$(\tau^*, \ddot{q}^*) = \underset{\tau, \ddot{q}}{\operatorname{argmin}} \; \|\ddot{q} - \ddot{q}^d\|^2$$

How to solve this?

s.t. $M\ddot{q} + h = \tau$

Joint position limits

* same approach to vel limits

$$q(t+\Delta t) = q(t) + \Delta t\, \dot{q}(t) + \frac{1}{2}\Delta t^2 \ddot{q}(t)$$

$$q_{min} \leq q + \Delta t \dot{q} + \frac{1}{2}\Delta t^2 \ddot{q} \leq q_{max}$$

- could result in large unfeasible accelerations!

Convex optimization problems

- have only one minimum → if local minimum found, it is the global minimum

Quadratic programs (QP)

* linear equality / inequality constraints
* convex quadratic cost: $\frac{1}{2} x^T H x + g^T x$, $H \succeq 0$

least squares programs (LSP)

subclass from (QP)

* linear equality / inequality constraints ($Ax \leq b$)
* cost is 2 norm of affine function $\|Ax - b\|_2$

QP in cartisan space

given reference e.e. trajectory $x^r(t)$, find $\tau(t)$

s.t. $x(t) \simeq x^r(t)$

$\ddot{x} = J\ddot{q} + \dot{J}\dot{q}$ $\quad \ddot{x}^d \triangleq \ddot{x}^r + K_p(x^r - x) + K_v(\dot{x}^r - x)$

$$\left.\begin{aligned} \ddot{q} &= J^\dagger(\ddot{x}^d - \dot{J}\dot{q}) \\ \tau^d &= M\ddot{q} + h \end{aligned}\right\}$$

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

$$(\tau^*, \ddot{q}^*) = \underset{\tau, \ddot{q}}{\operatorname{argmin}} \| J\ddot{q} + \dot{J}\dot{q} - \ddot{x}^d \|^2$$
$$\text{s.t } M\ddot{q} + h = \tau$$

Task Models

Generalizes e.e. control

* describes task w/ function $e(\cdot)$ to minimize
* e is error bt reference & real trajectory

$$e(x, u, t) = y(x, u) - \underbrace{y^r(t)}_{\text{trajectory}} \qquad x \triangleq (q, \dot{q}) \qquad u \triangleq \tau$$

cannot directly minimize (e) but we can its derivative in time (in the future) ← because it relies on q & $\dot{q}$ which are not decision variables in the LSP.

* 3 Kinds
  - Affine functions of $u \triangleq \tau$
  - Non-linear functions of $x \triangleq (q, \dot{q})$
* for func. of $\dot{q}$ ⇒ impose 1st derivative
* for func. of q ⇒ impose 2nd derivative

1. Velocity Task function

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

decision variable

$$\dot{e} = \frac{\partial y}{\partial q} \dot{q} - \dot{y}^r(t) = J\dot{q} - \dot{y}^r$$

$\underbrace{\frac{\partial q}{\partial t}}_{J}$ affine linear func

choose $\dot{e}$ as linear feedback:

$\dot{e} = -Ke \quad K > 0$

$J\ddot{q} - \dot{y}^r = -Ke \Rightarrow \boxed{\dot{e} + Ke = 0}$ Minimize

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

$Ax - b = 0$

2. Configuration Task function

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

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

$\dot{y}^r$ ← not a decesion variable

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

Choose $\ddot{e}$ as linear dynamics:

$\ddot{e} = -K_p e - K_j \dot{e}$

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

$\ddot{e} + K_p e + K_j \dot{e} = 0$ ← minimize

$$J\ddot{q} - (\ddot{y}^r - \dot{J}\dot{q} - K_p e - K_d \dot{e}) = 0$$

$Ax - b = 0$

Underactuated systems

- Using Task space inverse dynamics

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

s.t. ~~$[M \; -I]\, z = -h$~~

$[M \; -S^T]\, z = -h$ — Selector Matrix

Task space Inverse Dynamics Interactions

constrain the motion for rigid contact ! define it as a func

$c(q) = \text{const}$ ⇐ contact points don't move $J = \frac{dc}{dq}$
$J\dot{q} = 0$ ⇐ contact points velocities are null
$J\ddot{q} + \dot{J}\dot{q} = 0$ ⇐ contact points accelerations are null

Introduce force as a decesion variable :

$$\underset{(\ddot{q}, \tau, f)}{\text{argmin}} \|Az - b\|^2$$

s.t. $\begin{cases} M\ddot{q} + h = S^T\tau + J^T f \\ J\ddot{q} + \dot{J}\dot{q} = 0 \end{cases}$

Friction Cone Constraints

This is a quadratic constraint ⇒ can't use LSP

$\|f_T\|^2 \leq \mu f_N^2$ μ = friction coefficient

f_T: Tangential f_N: Normal

approximate w/ linear pyramidal cone: $Bf \leq 0$

Multi-Task control

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

$$\underset{z=(\ddot{q},\tau,f)}{\text{minimize}} \sum_{i=1}^{N} w_i g_i(z) = \|Az - b\|^2$$

$$\text{s.t} \begin{bmatrix} M & -J^T & -S^T \\ J & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ f \\ \tau \end{bmatrix} = \begin{bmatrix} -h \\ -\dot{J}\dot{q} \end{bmatrix}$$

Inverse Dynamics [Geometry]

*numerical I.G. ⇒ use computer program to find q by evaluating $F(\cdot)$ several times

↳ optimization problem

$$\underset{q}{\text{minimize}} \frac{1}{2} \|p(q) - p_d\|^2 = \frac{1}{2}\|e(q)\|^2 \triangleq C(q)$$

p_d = desired e-e position

$P(q)$ = Forward geometry function

1st order optimality conditions

Find where the gradient of the cost is zero

$$\frac{\partial}{\partial q} c(q) = 0 \Rightarrow \frac{\partial}{\partial q}\left(\frac{1}{2} e^T e\right) = e^T \frac{\partial e}{\partial q} = e^T J = \underbrace{J^T e}_{r(q)} = 0$$

J

$r(q) = 0$ ← start w/ guess $\bar{q}$

* Do linear expansion on cost $r(q)$

$$r(q + \Delta q) = r(\bar{q}) + \boxed{\nabla r(\bar{q})^T} \Delta q = 0$$ find this

$$\nabla r = \nabla (J^T e) = \underbrace{\nabla J^T e}_{\text{ignore}} + J^T \underbrace{\nabla e}_{J} \simeq J^T J$$

IF $N > 3$ ↑ not invertable

$$\therefore r(\bar{q}) + J^T J \Delta q = 0 \Rightarrow \Delta q = -(J^T J)^{-1} r(\bar{q})$$

Regularization linear expand e instead of cost

$$c(q) = \frac{1}{2}\|e\|^2 \simeq \frac{1}{2}\|\bar{e} + J\Delta q\|^2 = \frac{1}{2}\left[\bar{e}^T \bar{e} + \bar{e}^T J \Delta q + (J\Delta q)^T \bar{e} + (J\Delta q)^T J \Delta q\right]$$

$$= \frac{1}{2}\Delta q^T J^T J \Delta q + \bar{e}^T J \Delta q + \frac{1}{2}\bar{e}^T \bar{e} \Rightarrow \Delta q = -H^{-1} g$$

(H = $J^T J$, g^T = $\bar{e}^T J \Delta q$)

$$\underset{\Delta q}{\text{minimize}} \; \|e + J\Delta q\|^2 + \frac{\lambda}{2}\|\Delta q\|^2$$

regularization Cost

$$\Delta q = (J^T J + \lambda I)^{-1} J^T \bar{e} = -J^+ \bar{e}$$

Pseudo inverse

$\lambda > 0$ regularization weight ← very small

* Issues w/ vanilla gauss newton

1 - can get stuck @ zero slopes

2 - can get stuck in cycles

gradient only gives you local info, taking a large step won't guarantee cost goes down ← use line search

- I.G. step

- Input: $q^{(i)}$ (current guess), p^d output $q^{(i+1)}$ (next guess)

1. current error: $p(q^{(i)}) - p^d$
2. newton step: $\Delta q = -(J^T J + \lambda I)^{-1} J^T e$
3. line search: $q_{try} = q^{(i)} + \alpha \Delta q$, $\alpha = 1$

reduction $= \|e\| - \|e(q_{try})\|$

while reduction $< \gamma \alpha \|e\|$

$\alpha = \beta\alpha$; $q_{try} = q^{(i)} + \alpha \Delta q$; reduction

$q^{(i+1)} = q_{try}$

Optimal Control

- Can be used to do planning ⇒ no need for reference traj.
- Can handle constraints
- objective clearly specified by the cost function

optimal Control Problem (OCP)

- takes form of an optimization problem but it is not
  - ↳ $x(\cdot)$ & $u(\cdot)$ are not vectors ⇒ they are trajectories [infinite dimension objects]
  - ↳ the constraints are infinetly many because they all must be valid for all instances in range $[0, T]$

$$\underset{x(\cdot), u(\cdot)}{\text{Minimize}} \int_0^T \underbrace{l(x(t), u(t), t)}_{\text{running cost}}\, dt + \underbrace{l_f(x(T))}_{\text{Terminal cost}}$$

s.t.
- $\dot{x}(t) = f(x(t), u(t), t)$ Dynamics
- $x(0) = x_0$ initial conditions
- $g(x(t), u(t), t) \leq 0$ path | actuator constraints

OC Methods

	opt → Disc	Disc → opt
global	HJB	Dynamic Programing
local	Indirect Methods	Direct Methods

Dynamic Programming

- DP is a discrete time OCP
- Based on the principle of optimality:
  - the sub parts of an optimal trajectory is also optimal

DP approach

- Cut the minimization Problem into smaller minimization problems using Bellman's Principle

- $x(\cdot)$ & $u(\cdot)$ are trajectories in discrete time (matrices)
- Discrete time Dynamics

$$\underset{x,u}{\text{minimize}} \sum_{i=0}^{N-1} l(x_i, u_i)$$

$$\text{s.t. } x_{i+1} = f(x_i, u_i)$$

next state is a func of current state & control

→ Split into 2 & move the constraints into the cost of the function ← as an indicator function

Indicator func: 0 if input = 0; ∞ otherwise

makes constraints a very high cost if violated

$$\underset{U,x}{\text{Minimize}} \left[\sum_{i=0}^{M-1} \left(l(x_i, u_i) + I(f(x_i, u_i)) \right) + \sum_{i=M}^{N-1} \left(l(x_i, u_i) + I(f(x_i, u_i)) \right) \right]$$

I: indicator func; $f(x_i, u_i)$: dynamics constraint

$$\underset{x,u}{\text{Minimize}} \left[c_0(x_{1:M}, u_{0:M-1}) + c_M(x_M, x_{M+1:N}, u_{M:N-1}) \right]$$

separated because it is a parameter, not a value

$$\text{Minimize } c_0(x_{1:M}, u_{0:M-1}) + V_M(x_M)$$

optimal solution:

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

$V_M(x_M)$ ⇒ value function / optimal cost to go

optimal cost that we have to pay if we start from a state x_M onward behaving optimally.

DP Algorithm

- create the value function for last time step

$$V_N(x_N) = l_f(x_N)$$

- go backwards in time using recursive optimality

$$V_i(z) = \underset{u}{\text{minimize}}\ l(z, u) + V_{i+1}(f(z, u))$$

minimize only over one control value

- compute optimal control:

$$u_i^*(x) = \underset{u}{\text{argmin}}\ l(x, u) + V_{i+1}(f(x, u))$$

Q function in RL

- not just optimal control trajectory about also an optimal

feedback policy! ⇒ u is a func of x!

- result is a function [not a value] ⇒ Parametric optimiz.

Numerical Integration

Ordinary Differential equation (ODE)

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

- dependency of u has been omitted because u(t) is a function of t and/or x
- if f is lipshitz continous [first derivative of the func. is bounded] there exists a unique solution

1. Explicit Euler

- simplest numerical integration scheme

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

$$h\dot{x} \simeq x(t+h) - x(t) \Rightarrow x(t+h) = x(t) + hf(x, t)$$

$\dot{x}$

⇒ h must be sufficiently small & computationally exp.

2. Midpoint

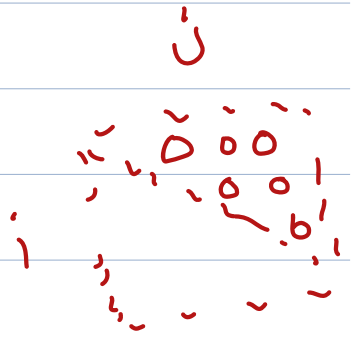
- Take a step to reach middle of time step
- Compute $\dot{x}$ there & use value for the whole step

$$K_1 = f(x_0, t_0) \quad , \quad K_2 = f\left(x_0 + \frac{h}{2} K_1, t_0 + \frac{h}{2}\right)$$

$$x_1 = x_0 + h K_2$$

- uses 2 function evaluations

3. Runge-Kutta (RK)



approx of $\dot{x}$

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

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

q = order of method

if $a_{ij} = 0 \ \forall j > i \Rightarrow$ explicit

- consistency order p: tells you how quickly the error goes to zero

---

Direct Methods

- Try to find a soln that is locally optimal
- Discretize OCP & use NLP to find local optimum solns
- Discretization steps:
  1. parameterize state & control w/ polynomials
  2. Enforce constraints based on a time grid

Methods:

1. Single Shooting (sequential) 3. Multiple shooting
2. Collocation (Simultanious)

1. Single Shooting

- it is about implementing a numerical integration scheme. The state is not a variable of the problem, but you need to compute it from the control [a decesion variable]
  → Integrate a differential equation which is the Dynamics of The system
- discretize control only $u(t)$ [piece wise constant]
  $u(t) = y_i \quad \forall t \in [t_i, t_{i+1}]$
- Compute $x(t)$ by integrating dynamics
- No need for dynamics constraint

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

$$\text{s.t. } g(x(t;y), u(t;y), t_i) \leq 0 \quad \text{path constraints}$$

- To compute the state & the running cost at the same time ⇒ we can use an augmented state!

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

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

integrate this instead of x & c separatly

Integrate the instead of $\dot{x} = f$

Sensitivities: the derivatives of the integration scheme (Gradient)

why? because we need the gradient of the cost function ⇒ which depends both on the state & control
However, the state also depends on the control.
dependency is decided by the integration scheme

cost func. using Euler

$$c(y) = \int_0^T l(x(t), y(t))\, dt + l_f(x(T))$$

$$\simeq \sum_{i=0}^{N-1} l(x_i, y_i) \cdot h + l_f(x_N) \quad \Leftarrow \text{differentiate}$$

$$\frac{dc}{dy} \simeq \sum_{i=0}^{N-1} \frac{\partial l(x_i, y_i)}{\partial y} \cdot h + \frac{\partial l_f(x_N)}{\partial y}$$

$$\frac{\partial l}{\partial x_i} \cdot \frac{\partial x_i}{\partial y} + \frac{\partial l}{\partial y} \qquad \frac{\partial l_f}{\partial x_N} \cdot \frac{\partial x_N}{\partial y}$$

depend on integration scheme

How to compute? Integration scheme

$$x_{i+1} = \Phi(x_i, y_i) \Rightarrow \frac{\partial x_{i+1}}{\partial y} = \frac{\partial \Phi_i}{\partial x_i} \cdot \frac{\partial x_i}{\partial y} + \frac{\partial \Phi}{\partial y}$$

- Initialize $\frac{\partial x_0}{\partial y} = 0$ because input affect future
- could use one of the Runge Kutta methods to calculate

Explicit Euler: $\bar{\Phi}(x, u) = x + hf(x, u)$

$$\frac{\partial \bar{\Phi}}{\partial x} = I + h\frac{\partial f}{\partial x} \qquad \frac{\partial \bar{\Phi}}{\partial u} = h\frac{\partial f}{\partial u}$$

2. Collocation

- discretize both $x(t)$ & $u(t)$
- enforce dynamics on a time grid

$$\left.\begin{aligned} x(t) &= s_i \\ u(t) &= y_i \end{aligned}\right] \forall t \in [t_i, t_{i+1}]$$

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

$$\frac{s_{i+1} - s_i}{t_{i+1} - t_i} - f\left(\frac{s_{i+1} + s_i}{2}, y_i\right) = 0$$

X approximation X approximation

→ Approximate cost integral

$$\int_{t_i}^{t_{i+1}} l(x(t), u(t))\,dt \simeq l\left(\frac{s_i + s_{i+1}}{2}, y_i\right)(t_{i+1} - t_i)$$

$$\triangleq l_i(s_i, s_{i+1}, y_i)$$

Problem formulation:

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

$$\text{s.t.} \quad s_0 - x_0 = 0 \quad \leftarrow \text{initial conditions}$$

$$c_i(s_i, s_{i+1}, y_i) = 0 \quad \leftarrow \text{dynamics}$$

$$g_i(s_i, y_i, t_i) \lesssim 0 \quad \leftarrow \text{actuator limits}$$

• VIP → this has a very important feature, which is sparsity! the gradient, the jacobian & the hessian of the problem [derivative of cost & const] are sparse matrices. Because they do not depend on many other variables! → only the i^{th} term

• use a solver that exploits sparsity.

3. Multiple Shooting (middle ground)

- discretize $u(t)$
- discretize $x(t)$ only @ a few points in time [course grid]
- Compute intermediate values of $x(t)$ by integrating dynamics

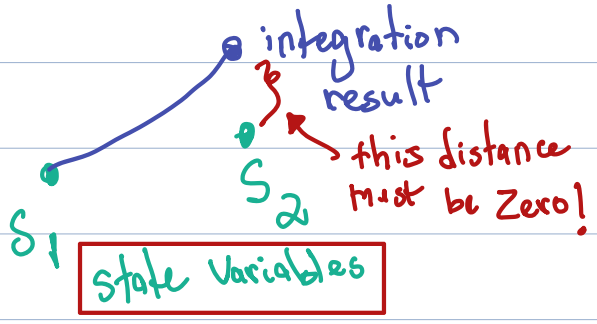
$$\left.\begin{aligned} u(t) &\approx y_i \\ \dot{x}(t) &= f(x_i(t), y_i) \end{aligned}\right] \forall t \in [t_i, t_{i+1}]$$

$x_i(t_i) = s_i$ ← few state variables

numerically compute integral:

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

VIP: Result of the numerical integration might not match w/ the value of the state variable @ the end
↳ continuity constraints must be implemented



→ problem formulation

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

$$\text{s.t.} \quad s_0 - x_0 = 0 \qquad g(s_i, y_i) \leq 0$$

→ continuity constraint ⇒

$$s_{i+1} - \underbrace{x_i(t_{i+1}, s_i, y_i)}_{\text{result of integration}} = 0$$

- Solver is allowed to violate that constraint while calculating partial solutions.
- Collocation & Multiple shooting find better solutions

Linear Quadratic Regulator (LQR)

- Method to solve a very specific type of OCP where the dynamics are a discrete time Linear system and a quadratic cost function
- Gives an efficent way of solving OCP by exploiting the structure of the problem and solving it as an LSP

$$x_{t+1} = \underbrace{A}_{\text{matrices}} x_t + \underbrace{B}_{\text{matrices}} u_t$$

objectives of LQR is to choose control u s.t. x is small & u is small.

↑ competing objectives

Problem formulation:

$$\underset{U}{\text{minimize}}\ J(U) = \sum_{i=0}^{N-1} \left(x_i^T Q x_i + u_i^T R u_i \right) + x_N^T Q_f x_N$$

quadratic form of

Q ⇒ state
R ⇒ control
Q_f ⇒ final cost

s.t. $x_{i+1} = A x_i + B u_i$

Q ↑ ⇒ small x more imp.
R ↑ ⇒ small u more imp.

Rewritten as follows:

$$\underset{(x,u)}{\text{minimize}}\ J(x,u) = \|\bar{Q} X\|^2 + \|\bar{R} U\|^2$$

Block diagonal mat.

s.t. $X = H x_0 + G U$

⇒ rewrite dynamics as linear eq:

$x_1 = Ax_0 + Bu_0 \Rightarrow x_2 = A^2 x_0 + AB u_0 + B u_1$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} A \\ \vdots \\ A^N \end{bmatrix} x_0 + \begin{bmatrix} B & \ddots & 0 & \cdots & 0 \\ AB & \ddots & B & \ddots & \\ \vdots & & & & \\ A^{N-1}B & & & & B \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

a linear func of state & control

- Remove x from the problem

$$\underset{U}{\text{minimize}} \; J(U) = \|\bar{Q}(Hx_0 + GU)\|^2 + \|\bar{R}U\|^2$$

$$= \|\bar{Q}GU + \bar{Q}Hx_0\|^2 + \|\bar{R}U\|$$

$$= \left\| \begin{bmatrix} \bar{Q}G \\ \bar{R} \end{bmatrix} U + \begin{bmatrix} \bar{Q}Hx_0 \\ 0 \end{bmatrix} \right\|^2$$

decision variable

$$\|Ax + b\|^2$$

LSP: is a 2 square norm of an affine function of a decision variable

LQR w/ DP

- LQR is a rare case where you can use DP because the parametric optimization problem is a quadratic program.

$$V_t(z) = \underset{u_t}{\text{minimize}} \sum_k \left(x_k^T Q x_k + u_k^T R u_k \right) + \left(x_N^T Q_f x_N \right)$$

↑ function of time & state

↑ optimal cost starting at state z @ time t

$V_0(x_0)$ = Optimal LQR cost

s.t. • $x_t = z$ • $x_{k+1} = A x_k + B u_k$

→ objective is to recursively minimize the value function starting @ the last time step & going backwards

Terminal cost of LQR: $V_N(z) = z_N^T Q_f z_N$

→ use belman principal to go backwards:

$$V_t(z) = \underset{w, u_{t+1}}{\text{minimize}} \; z^T Q z + w^T R w + \sum_{k=t+1}^{N-1} \left(x_k^T Q x_k + u_k^T R u_k \right) + x_N^T Q_f x_N$$

next step

s.t. $x_{k+1} = A x_k + B u_k \quad u_t = w$

$$V_t(z) = \underset{w}{\text{minimize}} \; z^T Q z + w^T R w + V_{t+1}(Az + Bw)$$

x_{t+1}

$V_{t+1}(z) = z^T P_{t+1} z \qquad P_n = Q_f$

$V_t(z) = z^T Q z + \min_w \left(w^T R w + (Az + Bw)^T P_{t+1} (Az + Bw) \right)$

$\Rightarrow \nabla f = 2Hw + 2g = 0$

soln = $w^* = -H^{-1} g$ — invertible by definition

+ve def $R + B^T P_{t+1} B$

classic soln for a QP [eg. IG, Newton]

$\therefore w^* = K_t z$ — gain matrix

Summary of LQR using DP:

1. you start by initializing your value function matrix P_N equal to terminal cost Q_f
2. you go back in time recursively to compute the matrix P_{t-1} starting from the matrix P_t
3. once you have P for all the time steps → (value function) you can compute the optimal control input for all the time steps by going forward in time $t = 0 \rightarrow t = N-1$
   - ↳ find the K_t matrix [gain]
4. once you have K_t → you can compute $u(t)$ → as it is a linear function $u_t = K_t x_t$

→ you alternate between using u_t to find x_{t+1} & multiplying $K_{t+1} * x_{t+1}$ to find u_{t+1}

* we recover an optimal feedback control policy

Differential Dynamic Programming

- non linear expansion of LQR
- handles non linear Dynamics & cost functions
- approach vaguely resembles Newton methods

Problem formulation

$$\text{minimize} \sum_{t=0}^{N-1} \ell_t(x_t, u_t) + \ell_N(x_N)$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t) \qquad x_0 = x^{init}$$

1. Start w/ initial guess for u
2. linearize around current trajectory
3. Solve LQR to get variation of V WRT previous guess
4. line search to ensure convergence

→ A heuristic, no guarantee of convergence

- Apply DP → But we can't apply to the original NL system. Instead we apply to a linearization [local approximation] of the system

$$V(z,i) = \min_w \left[l_i(z,w) + V(f(z,w), i+1) \right]$$

$$= \min_w Q(z,w)$$

- Take a quadratic approximation of Q function & minimize it.

→ DDP problem is reduced to minimizing the local quadratic model of Q

Soln: $w^* = \arg\min_w Q(z,w) = -\bar{Q}_{ww}^{-1}(Q_w + Q_{wx} z)$

- regularize
- Bias term
- affine func ⇒ not purely linear

→ more complicated than LQR because it is not purely linear func. of state, but affine.
Because cost is not purely Quadratic [you have the affine terms [linear]]

$$V = \Delta V + V_x z + \frac{1}{2} z^T V_{xx} z$$

const linear Quadratic

Summary of DDP

1. compute state trajectory $\bar{x}$ using initial guess for u going forward
2. Backwards pass ⇒ value func - terminal cost

then loop backwards in time to calculate:
- Derivatives of Q func.
- optimal control input w^*
- the Value func.

3. forward pass ⇒ don't have state yet → can't compute u

⇒ Do a line search starting w/ $\alpha = 1$
⇒ if cost didn't decrease enogh ⇒ take smaller step

Simulate w/ $u = \bar{u} + w$ by adding to the previous guess of $\bar{u}$

$$= \bar{u} + \alpha \bar{w} + K(z)$$
$$= \bar{u} + \alpha \bar{w} + K(x - \bar{x})$$

only applied to the feed forward part! not on feedback part.

differential dynamics

* assign $\bar{x} = x$, $\bar{u} = u$

* obtain optimal open loop control & a locally optimal linear feedback gain K

$$u = \bar{u} + K(x - \bar{x})$$

Regularization

tune regularization value automatically

↑ slow convergence

$\mu \downarrow$ • second derivative Q_{uu} singular
• $\bar{w}$ too large causes larger control inputs

if $\bar{Q}_{uu}$ not invertable or $\frac{J(\bar{U}) - J(U)}{\Delta J(\alpha)} < c_2$
then $\uparrow \mu$, otherwise decrease

Model Predictive Control (MPC)

- use OC for both computing & tracking trajectory
- seen this in DDP → gives feedback gains
- Solve a finite horizon OCP using current state as initial state

$$X^*, U^* = \underset{X,U}{\operatorname{argmin}} \sum_{k=0}^{N-1} l(x_k, u_k)$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k, k) \qquad x_0 = x_{meas}$$

Process:

1. Solve OCP
2. obtain optimal state & control trajectories
3. apply first value of control u_0 & throwout the rest
4. repeat for next iteration

* always behaving optimaly @ each step

- even w/o disturbance, the predicted trajectory using OCP & the one you get from MPC are going to be different!
  - ↳ Because reoptimizing uses a different time horizon in each step

Challenges: • feasibility • stability • Computation time

Infinite Horizon MPC

- all OCPs see the same (infinite) Horizon
  - ↳ will all be the same
  - if $N = \infty$ & cost $l(x,u) \geq \alpha \|x\|$, $\alpha > 0$
  - (1) having a finite cost implies stability
    - Because it can only be non-infinite if x goes to zero @ some point
  - (2) since predicted & actual are equal ⇒ I have recursive feasibility

- Soln: try to mimic an infinite horizon problem
  - * add terminal cost & terminal constraint

reminiscent of the value function in DP

Problem formulation:

$$\sum_{N-1}$$

$$V_N^*(\bar{x}) = \underset{U}{\text{minimize}} \sum_{k=0} l(x_k, u_k) + \underbrace{l_f(x_N)}_{\text{terminal cost}}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k) \quad | \quad x_0 = \bar{x}$$
$$x_k \in \mathcal{X}, \; u_k \in \mathcal{U} \quad | \quad \underbrace{x_N \in \mathcal{X}_f}_{\text{terminal constraint}}$$

Feasibility

two cases:

- Input constraints only: always feasible
- Hard state constraints: not just penalties
  - if $N < \infty \Rightarrow$ no guarantee for convergence

Soln #1: [MOAS] ← for linear systems

Use Maximum output admissible set theory

↳ $N < \infty$ is enough to enforce recursive feasibility

* however N must be sufficiently big

Soln #2: [MCIS]

Use Maximum Control Invariant Set

↳ can be used as a terminal constraint

- once you are inside the set you can stay inside the set

⇒ hard to compute for non linear systems

Suppose a system w/ linear dynamics

$$x^+ = Ax \qquad y = Cx$$

- have constraints on output [could be non linear]
- objective of algorithm ⇒ find a specific control invariant

set that is also the maximum output admissible set
↳ this the set of all initial states x, s.t. if you start in x the constraints on the output are always satisfied

General Stability theory [Lyapnov Stability]

exponential Lyapunov functions
↳ main tools to prove stability of nonlinear systems

- A set is positive invariant
  If x starts in S then it stays inside S

↳ Same as control Invariant sets w/o control
*autonomus dynamics ⇒ no u.

- if $\mathcal{X}$ is positive invariant ⇒ a value function V is an exponential lyapunov func if:

upper & lower bounds: $V(x) \geq \alpha_1 \|x\|$, $V(x) \leq \alpha_2 \|x\|$

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

↳ must decrease exponentially along trajectory

- the larger the norm of the state ⇐ the more decrease expected

Stability

- use Value function [optimal cost to go] as an exponential lyapnov function

$V_N^*(x) = \sum^{N-1} \ell(x_i, u_i) + l_f(x_N)$

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

Consider not optimal ⇒ conservative approach

⇒ difference between the value functions of two consecutive steps:

$$V_1(x_1) - V_0^*(x_0) = l(x_N, u_N) + l_f(x_{N+1}) - l(x_0, u_0) - l_f(x_N)$$

- if we can prove that this is $\leq \alpha_3 \|x_0\|$ then it is an exponential lyapnov func.

Stability Assumptions

1. origin is @equilibrium ⇒ runnig cost = 0
   ⇒ terminal cost = 0
2. union of state & control spaces is a closed set
   ⇒ terminal set is a compact set (closed, bounded)
3. for every state in terminal constraint set X_f, there must exist a value of u s.t. $f(x,u)$ belongs to X_f
   * X_f is control invariant
   * terminal cost decrease $l_f(f(x,u)) - l_f(x) \leq -l(x,u)$
     ↳ tricky to find an $l_f(x)$ that satisfies this condition
   * runnig cost must be lower bounded & terminal cost is upper bounded

Note: when we are looking for a local control lyapnov function ⇒ it is only valid inside a certain region.

* we expect that if we are sufficiently close to the origin, the optimal choice won't be affected by the constraints.

Summary:

- Ways to guarantee feasibility.

1. Pick $\mathcal{X}_f$ & $l_f(\cdot)$ s.t.
   - $\mathcal{X}_f$ is control invariant
   - $l_f(f(x,u)) - l_f(x) \leq -l(x,u)$ — at least as much as running cost
2. Can pick $\mathcal{X}_f$ as origin $= \{0\}$
   - ↳ origin is control invariant
   - ↳ terminal & running costs are zero } satisfy Pt #1
3. Can pick N large enough ⇒ no need for terminal cost or constraint ⇒ Brotforce w/ computation power

Computation Time

- To speed up computation:

– take the optimal soln computed @ time 0 & use it as an initial guess for the problem of next time step.

– instead of iterating until convergence ⇒ just do one optimization ⇒ insures you are doing the computation on the most recent information.

MPC uncertainties

- there will always be noise. 2 approaches

Robust MPC

– if you know the bounds of the noise ⇒ apply a safety

margin wrt the constraint boundary
- easier to implement & a lot more conservative

Stochastic MPC
- if you know the probability distribution of the noise
  - a lot less conservative

Reinforcement Learning

- A mathematical framework that allows for solving OCP
- Assume no knowledge of the system's mathematical model
- Tries to find the global optimal policy
- Assumes the dynamics to be stochastic
- Focus on optimization over an infinite horizon

Markov Decision Process

- Describes the enviroment in RL problem
- Defined by the tuple $<\mathcal{X}, \mathcal{U}, p, c, \gamma>$

→ **Markov property:** the future is independent of the past given the present.

→ **state transition Matrix:** a collection of all state transition probabilities ⇐ irreducable

→ **Policy:** Distribution of control inputs over the states. [stochastic] ⇐ assume deterministic

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

Markov Process/Chain: a tuple $< \mathcal{X}, P >$

$\mathcal{X}$: finite set of states P: state transition P matrix

- largest eigen value of P satisfies the following:

$$\min_i \sum_j P_{ij} < 1 \leq \max_i \sum_j P_{ij} \quad \text{(row)}$$

⇒ highest eigen value of the matrix is 1 & all eigen values of P are smaller than 1

* state evolution (Dynamics) uses probability density func $P(x^+|x)$

Markov Reward Process Defined as tuple $< \mathcal{X}, P, C, \gamma >$

C: cost func , γ: discount factor

- $\gamma \to 0$ myopic eval [future has small impact]
- $\gamma \to 1$ farsight eval [~ optimal control]

why? ⇒ • avoid inf. cost • uncertainty abt future

Bellman eq:

analytical form:

* $V(x) = \ell(x) + \gamma V(f(x))$ ⇐ for deterministic

matrix form:

* $V = C + \gamma P V$ ⇐ for both deterministic & stochastic

→ in stochastic systems ⇒ $f(x)$ becomes a Probability

distribution & cannot be used to calculate value func.

Action Value function (Q)

- **value func:** a func of the state & yields the ctg starting from that state & following a certain policy.
- **Action value func:** a func of state & control. yields the ctg starting from that state & applying that control & from that point onward follow a certain policy.

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

$$V^\pi(x) = Q(x, \pi(x)) \quad \text{1st control from the policy}$$

optimal value & Q functions

- choose the minimum value over all possible policies!

$$V^*(x) = \min_\pi V^\pi(x) \quad , \quad Q^*(x,u) = \min_\pi Q^\pi(x,u)$$

- one way to compute optimal policy:
  - ⇒ minimize optimal action value func Q^* over u!

$$\pi^*(x) = \underset{z \in U}{\operatorname{argmin}}\, Q^*(x,z)$$

missing info ⇒ Can't be calculated

$$= \underset{z \in U}{\operatorname{argmin}}\, l(x,u) + \gamma V(f(x,u))$$

Dynamics unknown

* if we know $Q^*(x,u)$ ⇒ use directly to calculate π^*

Bellman optimality eq:

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

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

→ non linear systems ← minimization is NL

Bellman operators

- **Bellman operator T^π:** $T^\pi(V) = C^\pi + \gamma P^\pi V$
  * input is a value function & output is another value function
- **Bellman optimality operator T:** $T(V) = \min_{u \in U} C^u + \gamma P^u V$
  * Do not have a policy to follow, you minize over all possible control inputs for all possible states.
  → it looks like the Bellman optimality principle

$$V_k(x) = \min_u l(x,u) + V_{k+1}(f(x,u))$$

Model Based Control: Dynamic Programing

- in DP, we assume full knowledge of the Model (MDP)
- 2 classes of problems:

1- Prediction * 1 algorithm

→ you have MDP, Policy π

→ need to find the value function of π

• Iterative Policy Evaluation

⇒ Apply Iterative Bellman expectation operator

$$V_{k+1} = l(x, \pi(x)) + \gamma V_k(f(x, \pi(x)))$$

$$\hookrightarrow V_{k+1} = T^{\pi} V_k \qquad k = \text{iteration \#}$$

- Compute a sequence of approximations of V_π where each approximation is computed from the previous approximation

⇒ Process:

1. start w/ arbitrary V_0
2. apply belman operator iteravley

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

3. guarntee to converge while $k \to \infty$

⇒ **proof:** Prove that V_π is a unique fixed point of T^π

⇒ V^π is the only point not effected by T^π

1. prove that T^π is contracting:

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

$$= \gamma \|P^\pi (V - Z)\|_\infty \qquad P^\pi \text{ only changes the order of vectors}$$

$$\leq \gamma \|V - Z\|_\infty \qquad \hookrightarrow \text{no change to max value}$$

2. Starting from V_0 ⇒ Prove that $V_k \to V_\pi$ as $k \to \infty$

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

$$\leq \gamma \|V_{k-1} - V_\pi\|_\infty \ \ldots\ldots$$

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

3. Prove V_π is unique fixed point of T^π

assume $T^\pi V = V$, $T^\pi Z = Z$

$\|T^\pi V - T^\pi Z\|_\infty \leq \gamma \|V - Z\|_\infty$ $\quad T^\pi$ is contracting

$\|T^\pi V - T^\pi Z\|_\infty = \|V - Z\|_\infty$ Both are fixed

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

2 soln: 1. $\gamma \geq 1$ ✗ impossible because $\gamma \in [0,1]$

2. $V = Z$ ✓ only holds when this is true

2. Control

* 2 algorithm

→ you have an MDP

→ need to find the optimal V^* & π^*

Policy Iteration ⇒ find optimal policy

Process:

(evaluate policy, improve policy)

1. start w/ arbitrary π_0
2. iterate for $k = 0 \to \infty$
   a. policy Evaluation $\pi_k \Rightarrow V^{\pi_k}$
   b. improve policy acting greedily wrt V^{π_k}

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

3. as $k \to \infty$ $\pi_k = \pi^*$

* **Proof:** By showing that each iteration of the algorithm you obtain a policy that is better than the previous one

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

one specific control

you have a bigger pool of controls

⇒ you can only be as bad or better

$$\min [Q^\pi(x,u)] \leq Q^\pi(x, \pi(x))$$

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

Modified policy iteration

- use m_k policy evaluation iterations to compute V^{π}
  - → if $m_k = \infty$ → policy iteration
  - → if $m_k = 1$ → value iteration

Value Iteration

- Compute optimal value function
- use to compute optimal policy

process:

1. start w/ arbitrary estimation of V_0
2. iterate for $k = 0 \to \infty$
   - update value func for all states

$$V_{k+1}(x) = \min_u \; l(x, u) + \gamma V_k(f(x, u))$$

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

3. guaranteed to converge as $k \to \infty$
4. use V^* to calculate $\pi^* = \arg\min_u l + \gamma V^*$

Model free prediction

No control yet

- we have a fixed policy & we just need to find the value function of that policy.
- vaguely resembles DDP approach

Monte Carlo (MC)

- Estimates V^π as the average cost to go

Process:

1. start from state you want to evaluate
2. run the policy
3. collect all costs untill the end of episode (must end)
4. sum all costs ⇒ mean estimate of value of state

→ because stochastic → need to take multiple samples

→ only applies to episodic MDPs

episode: one excution of policy from start to terminal state

total discounted cost $J_t = l_t + \gamma l_{t+1} + \gamma^2 l_{t+2} + \dots$

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

- in deterministic → 1 sample / no need for expected op

→ first_visit MC:

- increment only the first time you see a specific state. 1. counter ⇒ $N(x) \leftarrow N(x) + 1$

2. total cost ⇒ $C(x) \leftarrow C(x) + J(x)$

3. $V(x) = \frac{C(x)}{N(x)}$

→ every visit MC: counter & total cost are incremented every time a state is visited.

* incremental update: $V_N = V_{N-1} + \frac{1}{N}(J_N - V_{N-1})$

→ use a fixed value instead of N

$$V(x) \leftarrow V(x) + \alpha (J - V(x))$$

α ← how fast to update based on most recent sample

Temporal Difference Learning - TDO

- estimates the cost to go by using 1 step look ahead

$$V(x_t) \leftarrow V(x_t) + \alpha_t \left(\underbrace{\ell_t + \gamma V(x_{t+1})}_{\text{TD target} \Rightarrow \text{estimated ctg}} - V(x_t)\right)$$

TD target ⇒ estimated ctg — Based on my current estimation of value func

TD Error

MC: $V(x_t) \leftarrow V(x_t) + \alpha_t [J_t - V(x_t)]$

CTG $J_t \simeq \ell_t + \gamma V(x_{t+1})$ Bellman eq?

* Policy evaluation is TDO w/ $\alpha = 1$ and sampling all states uniformly

- TDO is stochastic approximation
- Since Bellman operator has a unique fixed point → TDO converges to V^π
- convergence guaranteed if α is [Robbins Monro]

MC vs. TDO

- MC is unbiased but has high variance
- TDO is biased but has low variance

TD(λ)

- use all the step-n cost to go algorithms & add them together using weights.

$$J_t^\lambda = (1-\lambda)\sum_{n=0}^{n-1} \lambda^{n-1} J_t^{(n)}$$

Forward View: $V(x_t) \leftarrow V(x_t) + \alpha\left[J_t^\lambda - V(x_t)\right]$

→ can only learn from complete episodes

Backward view:

- need to introduce Eligibility traces:

- measure how often & how recent you have visited state x

- $e(x)$ ↑ every time I visit x, otherwise it decreases

TD error $\delta_t = l_t + \gamma V(x_{t+1}) - V(x_t)$

$\therefore V(x) \leftarrow V(x) + \alpha\left[\gamma \delta_t e_t(x)\right]$

Model Free Control

	Value or Action Value	(value or action value) + Policy
MDP	$V(x)$ / $Q(x,u)$	$V(x)$ / $Q(x,u)$ + π
Known	Value Iteration	Policy iteration
unKnown	Direct Methods	Actor_Critic

- if MDP Known : learn value function V
- if MDP un_Known: learn Action Value function Q

→ $\pi(x) = \arg\min_u l(x,u) + \delta V(x')$

need MDP to calculate ↑

Mimize Q func instead

$\pi(x) = \arg\min Q(x,u)$

④ need input u that minimizes Q

***ON Policy learning:**

- follow the policy we want to learn
- learn on the job

***Off Policy learning:**

- control the system w/ a different policy
- look over someone's shoulder

min ← output

argmin ← input

Direct Methods: Q learning

• off policy method

- only learn Q function
- iteratively update an estimate of the Q function
- still use discrete state & control

→ keep track of $(x, u, l(x,u), x')$ and use them to improve Q based on Temporal Difference (TD)

$$\delta(Q) = l + \gamma \min_{u'} Q(x', u') - Q(x, u)$$

($\min_{u'}$: estimation of current)

• need to find lowest value of Q given u' as input

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

$$\mathbb{E}[\delta(Q)] = TQ - Q = 0$$

→ only one Q function can satisfy this expression → a unique fixed point of the bellman operator.

$$\therefore Q = Q^*$$

• needs exploration

Exploration Strategies

- Greedy → always choose control w/ best cost
- ε Greedy:
  * choose random control w/ probability ε
  * choose greedy control w/ probability $1-\varepsilon$
  * ε decreases over time → 0, explore @ beginning, exploit @ end
- Boltzmann exploration:
  - assign prob to each control proportional to its value
  - choose more often controls that leads to lower cost
- optimisim in the face of uncertainty:
  - Keep track of uncertainty of value of each control
  - decrease the uncertainty the more you sample that control
  - when you have a lot of uncertainty, you favor that control

Actor-Critic Methods: Generalized Policy Iteration

- Extension of Policy iteration to unknown MDP
- alternate bt [evaluating the policy & updating Q-function] & improving the policy using the value function you estimated

{ policy evaluation
  Policy improvement

- use Model free prediction Methods: MC, TD0, TD(λ)
- only use a rough eval of policy.

⇒ improve Policy based on approximate value func.

- on Policy

⇒ The policy used to generate transitions is not the same that is evaluated & improved

↳ the behaviour policy mixes exploration into the target policy

* No Gearantee that each iteration is better than the one before it.

Sarsa

policy evaluation ← critic

- exploit TD0 to learn Q from on policy samples
- use a ε greedy policy
- update estimate of Q function as follows:

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

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

→ very similar to Q learning → but no minimization of $Q(x',u')$!

→ if policy was fixed ⇒ same equation as [TD]

* as TD, SARSA can diverge in off-policy

Policy improvement ← Actor

options:

1- act greedly wRT Q ← computed on the fly

2- use ε-greedy as behaviour policy

3. Greedy in the limit w/ infinite exploration (GLIE)

→ ab is allowed not to be greedy in the limit, but as time → ∞ the police becomes greedy

* all state–control pairs are visited infinitly many times
* Policy converges to a greedy policy
* Robbins Monroe condition for step size

Summary

	Policy Eval	Policy Iter.	Value iteration
DP	$V(x) \leftarrow l + \gamma V(x')$	$V(x) \leftarrow l + \gamma V(x')$ $\pi(x) \leftarrow \arg\min_u l + \gamma V(x'(u))$	$V(x) = \min_u l + \gamma V(x'(u))$
TD	TD learning $V(x) \leftarrow V(x) + \alpha \delta$	Sarsa $Q(x,u) = Q(x,u) + \alpha \delta(Q)$ $\pi(x) = \arg\min_u Q(x,u)$	Q-learning $Q(x,u) = Q(x,u) + \alpha \delta(Q)$

Value function Approximation

- Discrete space algorithms don't scale well for large systems
- use function approximation techniques instead of tables to represent V or Q

$$\hat{V}(x, w) \simeq V_\pi(x) \qquad \tilde{Q}(x, u, w) \simeq Q_\pi(x, u)$$

- w is the weights/parameters of the function

→ this is just a parametric function

↳ the learning process now aims to learn those parameters s.t. the function approximator

reseambles the actual one
⇒ reduces the size of the vector you're work w/
→ Any differential approximator can be used:
- in RL, FA are usually NNs
- linear combinations . polynomials . Fourier basis
⇒ as long as it is parametirized & differntiable

problems:
1. trying to approximate a moving target
2. The data are not independently normally distributed [iid]

Gradient Descent
→ most common approach
→ take a small step in the direction of gradient of function that you want to minimize

$$\min_w J(w) \Rightarrow \quad w = w - \frac{1}{2} \alpha \nabla_w J(w)$$

$\alpha \Rightarrow$ step size

- if step is sufficiently small → converge to a local opt.

Stochastic Gradient Descent (SGD)
- it is stochastic because the quantity we are Trying to minimize is the expectation of the squqre difference bt the real value function & the approximated one.
- it is an expectation because the value of this

depends on the policy π ⇒ we want to take the expectation of u following the distribution of π

$$J_w = \mathbb{E}\left[(V_\pi(x) - \hat{V}(x,w))^2\right]$$
$$\Delta w = -\frac{1}{2}\alpha \nabla_w J = \alpha\left[(V_\pi - \hat{V})\nabla_w \hat{V}\right]$$

Process:

- Run a simulation following a policy π that you want to evaluate.
- if we perform the update by sampling the gradient we want to minimize ⇒ then the expected update is equal to the full gradient update that we would get by calculating the gradient of the expectation.

Incremental prediction algorithms

- need a target for our value function.
- for MC: compute cost to go $J_t(x)$

$$\Delta w = \alpha(J_t - \hat{V}(x_t, w))\nabla_w \hat{V}(x_t, w)$$

⇒ converges to a local optimum with NL
⇒ in linear approximators ⇒ problem is convex.

For TD(0): Target given by TD target:

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

⇒ converges close to a global optimum with linear

→ in NL approximators ← not guaranteed because of Bias.

Increment control w/ function approximation

- need to switch from value func to Q because we don't know the dynamics.

→ DO generalized policy Iteration (GPI) w/ an approximate action value func: (Actor_critic)

$$J(w) = \min_w \mathbb{E}_\pi \left[\left(\hat{Q}(x,u,w) - Q_\pi(x,u) \right)^2 \right]$$

→ SGD to find local minimum:

- MC: target is ctg $\Rightarrow J_t(x_t, u_t)$
- TDO: target is $\ell_t(x_t,u_t) + \gamma \hat{Q}(x_{t+1}, u_{t+1}, w)$
- use ε greedy to ensure exploration

Batch learning

- Learn only after having collected a batch of data
- use the batch to perform an update on ws of nn

$$\min_w \sum_{t=1}^{T} \left(v_t - \hat{v}(x_t, w) \right)^2$$

- Can be done using experience replay buffer

Deep Q Network

- an excution of Q learning to continous state space & discrete control
- uses experience replay buffer
- use SGD to optimize w

* use fixed Q_targets :
  - use one nn for computing the target
  - use another as the main Q and update its ws
  - optimize MSE between the 2 networks

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