Computational Control - FS 2024

Aschari Eric, Mastroddi Giacomo, Muttoni Marco

August 18, 2024

Contents

1 Other Controllers

1.1 PID

Pros

- Simple
- Known tuning methods
- Low computations

Cons

- Reactive response
- SISO (and some MIMO with cascaded)
- Nonlinear

Examples

- Reference tracking
- Position control drones
- Flow, temperature heating, chemical

1.2 Frequency Domain Design

Pros

- Stability analysis (e.g., Nyquist, Bode plots)
- Robustness to model uncertainties
- Effective in handling disturbances

Cons

- Complex design process
- Limited applicability for nonlinear systems: requires linear models

Examples

- Control of power systems (stability)
- Vibration analysis in mechanical systems
- Filter design in communication systems

2 Linear Quadratic Regulator LQR

Pros

- Optimal control law
- Guaranteed stability of the closed-loop system
- Online computationally less expensive
- Ideal for linear systems
- Flexibility with tuning weights

Cons

- Linear system model
- Model accuracy
- Non-quadratic cost
- Constraints
- Robustness is limited
- Large memory for finite horizon

Examples

- Aircraft autopilot
- Path tracking mobile robots
- Infinite horizon: persistent tracking in process automation, satellite trajectory

In general, optimization problems are typically intractable:

- size of u is too large.
- require an accurate model of the system.
- non-convex (between two solutions you don't get another solution).
- open-loop nature make them useless in the presence of disturbances.

Finite Horizon

$\Rightarrow$ returns in a sequence of time-varying gain matrices.
Tractable optimization problem: Linear dynamics and Quadratic cost.

- **Markovian state** representation: $\begin{cases} \text{Next state is linearly dependent only on current state: } x_{t+1} = Ax_t + Bu_t. \\ \text{Quadratic cost is additive over time: } \sum\limits_t x_t^\top Q x_t + u_t^\top R u_t. \\ \text{Initial condition } x_0 \text{ is known.} \end{cases}$

- The problem is convex and decomposable $\rightarrow$ solvable via **backward induction** (**Bellman Optimality Principle**: An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.)

$$V_t(x) = \min_{u_t,\dots,u_{T-1}} \sum_{s=t}^{T-1} \underbrace{(x_s^\top Q x_s + u_s^\top R u_s)}_{\text{running cost}} + \underbrace{x_T^\top S x_T}_{\text{terminal cost}}$$
$$\text{s.t.} \quad x_{s+1} = Ax_s + Bu_s, \quad s = t,\dots,T-1$$
$$x_t = x$$

where the optimal control sequence is given by

$$u_t = \underbrace{-(R + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A}_{\Gamma_t} x_t$$
$$P_{t-1} = Q + A^\top P_t A - A^\top P_t B (R + B^\top P_t B)^{-1} B^\top P_t A, \quad P_T = S$$

Considerations

- $V_t(x)$ is the value function.
- $V_t(x) = x^\top P_t x$ is quadratic, i.e $P \succeq 0$ and symmetric.
- $V_0(x_0)$ is the (total) optimal cost.
- The result is a linear feedback control law, i.e accounts for disturbances.
- **Offline computation**
  - $n \times n$ matrix multiplications (A)
  - $m \times m$ matrix inversion (R)
  - T iterations (horizon)

Online computation

- Storage of T $m \times n$ matrices (Γ)
- $m \times n$ matrix multiplications ($\Gamma_t x_t$)

Infinite Horizon

$\Rightarrow$ returns a constant gain matrix.
Extension of the LQR to an infinite horizon $T = \infty$ for LTI persistent tracking and regulation problems.

- The iteration

  $$P_{t-1} = Q + A^\top P_t A - A^\top P_t B(R + B^\top P_t B)^{-1} B^\top P_t A, \quad P_0 = 0$$

  converges, as $t \to \infty$, to a solution $P_\infty \succ 0$ of the Algebraic Riccati Equation (ARE):

  $$P = Q + A^\top PA - A^\top PB(R + B^\top PB)^{-1} B^\top PA.$$

  where the optimal feedback control is given by

  $$u_t = \underbrace{-(R + B^\top P_\infty B)^{-1} B^\top P_\infty A}_{\Gamma_\infty} x_t$$

Considerations

- The sequence $P_0, P_{-1}, P_{-2}, \dots$ is non-decreasing due to Bellman Optimality Principle.
- The sequence $P_0, P_{-1}, P_{-2}, \dots$ is upper-bounded because it converges towards zero.
- (A, B) stabilizable and $(Q^{\frac{1}{2}}, A)$ detectable then ARE has a unique $P_\infty \succeq 0$ and $A + BK_\infty$ is asy. stable.
- In practice: unstable modes need to be weighted in the cost function.
- **Offline computation**
  - $n \times n$ matrix multiplications (A)
  - $m \times m$ matrix inversion (R)
  - "∞" iterations (horizon)

Online computation
  - Storage of one $m \times n$ matrix (less memory required!)
  - $m \times n$ matrix multiplications $(\Gamma_t x_t)$

3 Model Predictive Control MPC

3.1 MPC and Tracking MPC

Pros

- Optimal control law
- Guaranteed stability
- Constraints
- Disturbance rejection via receding horizon
- LTI: parametrized OCP in x_0
- Markovian systems

Cons

- Linear system model
- Model accuracy
- Computationally expensive & need to happen in one time step (online)
- Discrete state space
- Steady-state error
- Input disturbances, process noise, multiplicative noise

Examples

- Reference tracking
- Cruise control, autonomous driving
- Planning tasks

It is a static (memoryless), nonlinear (non-continuous), time-invariant feedback control law.
It is computed by solving an optimization control problem (OCP) parametrized by the initial state x_0 (time-invariant) and uses a receding horizon (same horizon length).

$$\begin{aligned}
\min_{u,x} \quad & \sum_{k=0}^{K-1} g_k(x_k,u_k) + g_K(x_K) \\
\text{subject to} \quad & x_{k+1} = f(x_k,u_k), && k=0,\dots,K-1 \\
& x_0 = x \\
& x_k \in \mathcal{X}_k, && k=0,\dots,K \\
& u_k \in \mathcal{U}_k, && k=0,\dots,K-1 \\
& x(T) = x_{\text{terminal}}
\end{aligned}$$

Considerations

- Does not manage to solve every optimization problem (for example when the optimal input turns out to be a the end of the horizon and there is no terminal constraint).
- **Offline computation of $u_0^\star(.)$**
  - Extremely complex
  - Abundant computation power
  - Large memory requirement (store all trajectories)
  - Example: unconstrained linear MPC: first element of finite-time LQR

  Online computation of $u_0^\star(.)$
  - Tractable
  - Limited time
- Tracking is achieved by a change of coordinates $\begin{cases}\tilde{x} = x - x_{\text{ss}} \\ \tilde{u} = u - u_{\text{ss}}\end{cases}$
  - Change of variables is inconsequential for LTI systems.
  - x_{ss} is derived by first principles or from offline **steady state optimization** computation.

Finite Horizon

Discrete time system $x(k+1) = Ax(k) + Bu(k)$ equilibrium point are defined by fixed points $x^\star = Ax^\star + Bu^\star$.

Lyapunov Stability for Discrete-Time

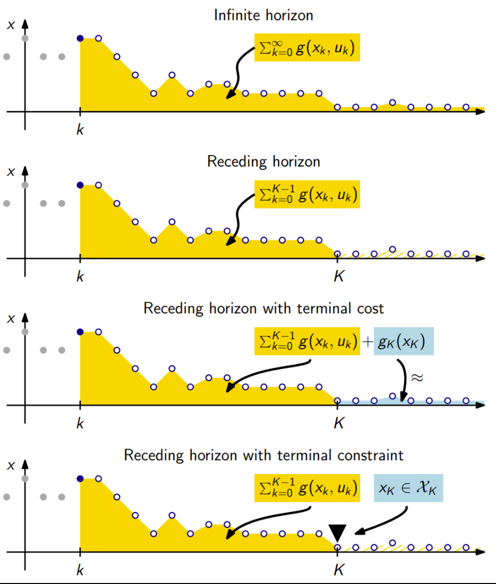
Let $x^\star = 0$ and V be a continuous function such that

- $V(0) = 0$ and $V(x) > 0,\ \forall x \neq 0$
- $\Delta V(x(k)) \triangleq V(f(x(k))) - V(x(k)) < 0,\ \forall x(k) \in \mathcal{D}$
- V radially unbounded: $\|x\| \to \infty \Rightarrow V(x) \to \infty$

$\Rightarrow$ then the origin is **globally asy. stable**.

Terminal constraint: $x(T) = x_{\textbf{terminal}}$

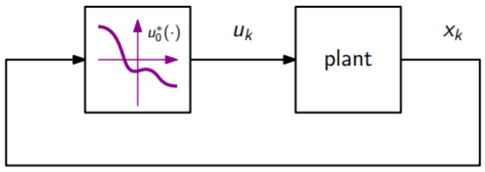
$$
\begin{aligned}
W(x_{k+1}) &= \min \sum_{s=k+1}^{k+K} g(x_{ss}, u_{ss}) \\
&= \min \sum_{s=k}^{k+K-1} g(x_{ss}, u_{ss}) - g(x_k, u_k) + \cancel{g(x_{k+K}, u_{k+K})}) \\
W(x_{k+1}) &\le W(x_k) - g(\tilde{x}_k, \tilde{u}_k) \le W(x_k)
\end{aligned}
$$

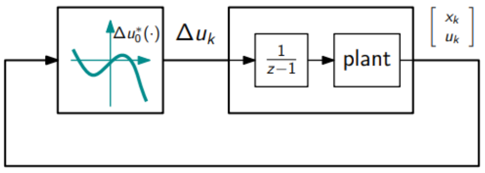


Considerations

- Stability can be lost when receding horizon is too short or when full state is not available.
- Other sufficient conditions to guarantee MPC stability
  - Terminal cost function: $\phi(x(T))$
  - Terminal constraint set: $x(T) \in \mathcal{X}_{\text{terminal}}$
  - Large receding horizon K

3.1.1 Incremental MPC





Problems:

- MPC will **not** achieve $x_{ss} = 0$ if a non-zero steady-state input is needed, because it inherently minimizes the input effort.
- Fragile against input disturbances, process noise, multiplicative noise.
- Low input can cause numerical issues.

Solutions:

- **Robustness**: Augment the plant with input integrator $\frac{1}{z-1}$ (if not present): more robust against disturbances and noise.
- **Numerical stability**: By penalizing $\Delta u_k = u_{k+1} - u_k$.

3.2 Robust MPC

Pros

- Noise rejection ensuring feasible constraints
- Uncertainties within model

Cons

- Info about the noise/disturbance
- Open-loop: too conservative
- Closed-loop: difficult to find a parametrization

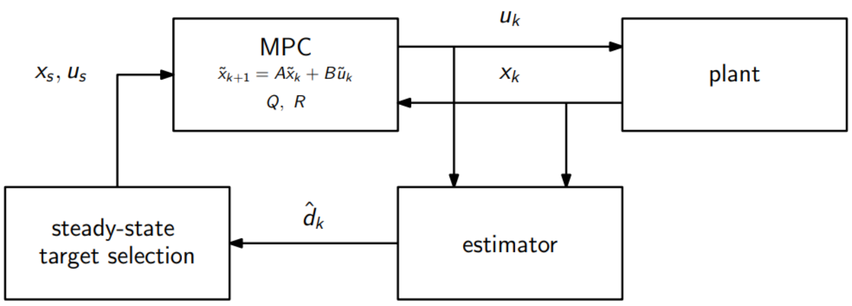
Examples

- Feasibility & robustness: aerospace
- Reliability & safety: driver assistant
- Power grids

Problems:

- Cannot achieve zero tracking error due to the lack of an integrator.
- Neglected exogenous disturbances: $x_{k+1} = f(x_k, u_k, w_k)$
- Model mismatch: $x_{k+1} = \tilde{f}(x_k, u_k)$
- Missing dynamics / non-Markovianity: $\begin{cases} x_{k+1} = f(x_k, z_k, u_k) \\ z_{k+1} = f_z(x_k, z_k, u_k) \end{cases}$
- Linearization: $x_{k+1} = \tilde{A}x_k + \tilde{B}u_k$
- And more: time discretization, quantization, time-varying parameters

3.2.1 Disturbance Rejection in MPC


MPC
$\tilde{x}_{k+1} = A\tilde{x}_k + B\tilde{u}_k$
$Q,\ R$
u_k
x_k
plant
x_s, u_s
steady-state target selection
$\hat{d}_k$
estimator

Solutions:

- Model the disturbance as constant: $\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix} = \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} x_k \\ d_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k$
- Predict of x_{k+1} based on measurements x_k, u_k and estimate $\hat{d}_k$: $\hat{x}_{k+1} = Ax_k + Bu_k + B_d\hat{d}_k$
- Correct based on the prediction error: $\hat{d}_{k+1} = \hat{d}_k + L(x_k - \hat{x}_k)$
- Select steady-state closer to specifications:

$$\begin{aligned} \min_{x_{ss}, u_{ss}} \quad & \|x_{ss} - x_{spec}\|^2_{Q_{ss}} + \|u_{ss} - u_{spec}\|^2_{R_{ss}} \\ \text{subject to} \quad & \begin{bmatrix} I - A & -B \end{bmatrix} \begin{bmatrix} x_{ss} \\ u_{ss} \end{bmatrix} = B_d\hat{d}_k \\ & x_{ss} \in \mathcal{X} \\ & u_{ss} \in \mathcal{U} \end{aligned}$$

3.2.2 Robust MPC

Closed-loop solution:

- Constructs a **feedback control** $u_1(x_1), \dots, u_K^\star(x_K)$
- Computationally intractable (except for soft-constrained min-max LQR: worst-case scenario)
- Optimal: all past information about the disturbance is used at each stage k

$$\hat{w}_k = D^\dagger(x_{k+1} - Ax_k - Bu_k)$$

- **Dynamic programming** automatically returns the desired control law $u_0^\star(x)$ from offline computation.
- Corresponds to infinite-time optimal control at the limit $K \to \infty$.

Open-loop solution:

- Constructs an **input sequence** $u_0, u_1, \dots, u_K$
- Computationally tractable (convex optimization problem) but often **unfeasible**
- Too conservative: no past information about the disturbance is used except for the information available at $k = 0$
- The desired control law $u_0^\star(x)$ is obtained by parametrizing the online optimization problem in x

3.2.3 Feedback MPC

A trade-off of the two solutions above: affine control law $\boxed{u_k = v_k + Lx_k}$

- v_k: Open-loop (feedforward) sequence/optimal policy
- L: Closed-loop (feedback) gain: rejects disturbances: $x_{k+1} = (A + BL)x_k + Bv_k + Dw_k$
  - $A + BL$ Hurwitz
  - L designed offline (can use Riccati by relaxing constraints of MPC)

Alternative: optimize both v and L online $\to$ computationally complex $\to$ Tube MPC.

3.3 Economic MPC EMPC

Pros

- Considers complex metrics
- Efficient trajectories (transients)
- Linear program for linear ℓ
- Future feasibility with steady-state terminal constraint

Cons

- More difficult to solve (not convex)
- Balancing economic objectives requires trade-offs
- Stability not guaranteed

Examples

- Efficiency tasks
- Processes: furnace, chemical industry
- Logistics, supply chain

It considers a **continuous** and **lower bounded** stage cost $\ell(x,u)$ (e.g. linear functions) that represents economic losses, energy use, cost of material, ...

$$\begin{aligned}
\min_{u,x} \quad & \sum_{k=0}^{K-1} \ell(x_k,u_k) \\
\text{subject to} \quad & x_{k+1} = f(x_k,u_k), \quad k=0,\dots,K-1 \\
& x_0 = x \\
& x_k \in \mathcal{X}_k, \quad k=0,\dots,K \\
& u_k \in \mathcal{U}_k, \quad k=0,\dots,K-1
\end{aligned}$$

Alternative: **Steady-state optimization** problem and **tracking MPC** to follow reference:

Advantages

- Easier to use $\ell(x,u)$

Disadvantages

- Suboptimal solution.
- Cost is not reduced during system transients.
- Does not respond efficiently to disturbances. (Traffic example: offline compute x_{ss}, u_{ss}, but if there's an incident, they do not adapt to the new situation.)

Infinite Horizon

- Economic interpretation (continuous processes, long-term gains, ...).
- May not drive the system to an equilibrium (not necessarily a problem).
- Boundedness is guaranteed by the constraints (feasible future trajectory).
- Computationally intense due to infinite horizon.

Finite Horizon

- Computationally tractable.
- Unstable trajectories can emerge: cheap trajectory now, expensive fix later.
- Infeasibility at the end of the control horizon.

Terminal constraint: (x_{ss}, u_{ss})

- Computationally more tractable.
- Future feasibility.
- Stability is **not guaranteed** as it was for MPC because $\ell(x,u)$ is not minimized at steady state. Nonetheless, asymptotic stability is **not desired** when using EMPC, such that it produces trajectories that are **efficient and economical** (also limit cycles/periodic orbits) and do not converge to an equilibrium, which exhibits a higher cost.
- Even in the case in which the system converges to the steady state, EMPC produces **efficient** trajectories to the equilibrium (theorem).

Note: stronger conditions are required to guarantee asy. stability.

4 System ID & Data-enabled Predictive Control DeePC

Pros

- Model-free
- More robust on noise
- No state estimator needed
- Many uses for predictor: filtering, control, recover missing data, prediction/simulation

Cons

- Computationally complex: $2n + K$ decision variables
- Larger memory footprint: data
- Based on LTI systems
- Mobel-based better if a state is known
- SysID incorporates available system info
- Hard and expensive to get data

Examples

- Temperature regulation
- Smart grid
- Medical devices
- Trading, price prediction and optimization

4.1 System Identification SysID

Goal: find **minimal** (controllable & observable) **realization** (state-space representation) given input-output data (not unique!).

$$x_k = \underbrace{A^k x_0}_{\text{free response}} + \underbrace{\sum_{j=0}^{k-1} A^{k-1-j} B u_j + \sum_{i=0}^{k-1} A^{k-1-i} D_w w_i}_{\text{convolution}}$$

$$y_k = C x_k + \underbrace{D u_k}_{\text{feedthrough}}$$

- For a LTI, the impulse response is a complete representation of the input-output relation, as we can combine responses (superposition).
- **Markov Parameters**: sequence of $CA^{k-1}B$ $\in \mathbb{R}^{p\times m}$ $k > 0$ where each element represent the output i to an unit impulse on input j.
- At least m impulse experiments are necessary to estimate them (needed to get matrix B).

4.1.1 Reachability - Controllability

Investigate all states that can be reached at time τ starting at $x(0) = 0$.

$$\mathcal{C}_n = [B\ AB\ A^2B\ A^3B\ \dots\ A^{n-1}B] \quad \in \mathbb{R}^{n\times n} \qquad \text{Reachable: full row rank} \Leftrightarrow \text{rank}(\mathcal{C}_n) = n$$

- For linear-continuous time: set of reachable states = set of controllable states. System is completely reachable $\Rightarrow$ System completely controllable.
- Differences between controllability and reachability:
  - **Compl. controllable**: may be kept to the origin, starting at any initial condition.
  - **Compl. reachable**: can only be brought to a certain point $x(T)$, but cannot be forced to stay there for $t > T$.

Considerations

- Scalar input creates column vector matrices $A, AB, \dots \Rightarrow$ need n steps for full rank
- Compl. controllable in less than n, if input u is **not** scalar.
- Not compl. controllable in n steps $\Rightarrow$ more steps won't help (linearly dep. on previous ones)
- Input sequence to reach target state $x_n = \mathcal{C}_n \begin{bmatrix} u_{n-1} \\ \vdots \\ u_0 \end{bmatrix}$ if $\begin{cases} \text{invertible} & \mathcal{C}_n^{-1} \\ \text{non-invertible} & \mathcal{C}_n^T(\mathcal{C}_n\mathcal{C}_n^T)^{-1} \quad \text{(smallest norm)} \end{cases}$

4.1.2 Observability

Reconstruct the initial condition $x(0)$ by analyzing only the output $y(t)$.

$$\mathcal{O}_n = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad \in \mathbb{R}^{n\times n} \qquad \text{Observable: full column rank} \Leftrightarrow \text{rank}(\mathcal{O}_n) = n$$

Considerations

- Scalar input creates row vector matrices $C, CA, \dots \Rightarrow$ need n steps for full rank
- Compl. observable in less than n, if input u is **not** scalar
- Not observable in n steps $\Rightarrow$ more steps won't help (linearly dep. on previous ones)
- Initial state to produce target output sequence $\begin{bmatrix} y_0 \\ \vdots \\ y_{L-1} \end{bmatrix} = \mathcal{O}_L x_0$ if $\begin{cases} L < n \text{ non-inv.} & \mathcal{O}_L^T(\mathcal{O}_L\mathcal{O}_L^T)^{-1} \quad \text{(smallest norm)} \\ L = n \text{ invertible} & \mathcal{O}_n^{-1} \\ L > n \text{ overdet.} & \text{not all } y \text{ are valid} \end{cases}$

4.1.3 Hankel Matrix

Hankel matrix composed of Markov parameters: $\mathcal{H} = \mathcal{O}\mathcal{C}$
Ho-Kalman algorithm (extended):

1. Construct $\mathcal{H}_{k,l}$:
   - Collect impulse responses as Markov parameters
   - Estimate $k, l > n$: stop adding Markov parameters once rank n is reached (rank $\mathcal{O}$ and $\mathcal{C}$ is n such that they are compl. observable and controllable and automatically rank $\mathcal{H}$ is n)
2. Factorize $\mathcal{H}_{k,l}$ in $\mathcal{O}_k\mathcal{C}_l$:
   - SVD: $\mathcal{H}_{k,\ell} = [\mathbf{U}_1 \quad \mathbf{U}_2] \begin{bmatrix} \Sigma_1 & 0_{r\times n-r} \\ 0_{m-r\times r} & 0_{m-r\times n-r} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \end{bmatrix}^\top = \mathbf{U}_1\Sigma_1\mathbf{V}_1^\top \qquad U \in \mathbb{R}^{m\times m}, V \in \mathbb{R}^{n\times n}$
   - $\mathcal{H}_{k,l} = \underbrace{\mathbf{U}_1\Sigma_1^{1/2}}_{\mathcal{O}_k}\underbrace{\Sigma_1^{1/2}\mathbf{V}_1^\top}_{\mathcal{C}_l}$
   - Shift $\mathcal{H}_{k,l}^\uparrow$ to get A
3. Derive A, B, C: $\quad A = \mathcal{O}_k^\dagger \mathcal{H}_{k,l}^\uparrow \mathcal{C}_l^\dagger \qquad B$ from $\mathcal{C}_l \qquad C$ from $\mathcal{O}_k$

4.2 Behavioral Representation

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{L-1} \end{bmatrix} = \underbrace{\begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{L-1} \end{bmatrix}}_{\mathcal{O}_L \in \mathbb{R}^{Lp\times n}} x_0 + \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ CB & 0 & \cdots & 0 \\ CAB & CB & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{L-2}B & CA^{L-3}B & \cdots & CB & 0 \end{bmatrix}}_{\mathcal{G}_L \in \mathbb{R}^{Lp\times Lm}} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{L-1} \end{bmatrix}$$

$\mathcal{O}_L$: extended observability matrix (free evolution of the system)
$\mathcal{G}_L$: convolution matrix (forced evolution of the system)

- $\mathbf{y}_L - \mathcal{G}_L\mathbf{u}_L = \mathcal{O}_L x_0$ solved if $\mathbf{y}_L - \mathcal{G}_L\mathbf{u}_L$ is in the **column image** of $\mathcal{O}_L$: $\begin{cases} Lm < n & \text{underdetermined} \\ Lm = n & \text{invertible} : x_0 = \mathcal{O}_n^{-1}(\mathbf{y}_n - \mathcal{G}_n\mathbf{u}_n) \\ Lm > n & \text{overdetermined} \end{cases}$
- for $Lm > n$: $\mathbf{y}_L - \mathcal{G}_L\mathbf{u}_L$ needs to belong to a **subspace** spanned by $\mathcal{O}_L$
- All I/O trajectories $(\mathbf{u}_L, \mathbf{y}_L)$ belong to the **column image** of Λ_L / a **subspace** of dimension $Lm + n$: $\begin{bmatrix} \mathbf{u}_L \\ \mathbf{y}_L \end{bmatrix} = \underbrace{\begin{bmatrix} I_{Lm} & \mathbf{0}_{Lm\times n} \\ \mathcal{G}_L & \mathcal{O}_L \end{bmatrix}}_{\Lambda_L \in \mathbb{R}^{L(p+m)\times Lm+n}} \begin{bmatrix} \mathbf{u}_L \\ x_0 \end{bmatrix} = \mathcal{H}\mathbf{g}$
- Λ_L fully describes the system (I/O sequence compatible with the plant $= x_0$ such that ...)

$\rightarrow$ columns of Λ_L are a basis of the subspace.
$\rightarrow$ columns of $\mathcal{H}$ are a basis of the subspace, with $\mathbf{g} \in \mathbb{R}^{Lm+n}$ being a vector of coefficients.

4.2.1 Relations

- $\boxed{T_{\text{ini}} = T_{\text{past}}}$ length of the past data points used to initialize the problem $\begin{cases} \text{For SISO:} & T_{\text{ini}} \geq n \\ \text{For MIMO:} & T_{\text{ini}} \geq \ell \text{ (lag } \ell \leq n) \end{cases}$
- $\boxed{T_{\text{fut}} = K}$ prediction horizon or the length of the future trajectory.
- $\boxed{L = T_{\text{ini}} + T_{\text{fut}}}$ total length of the trajectory.
- $\boxed{T \geq (m+1) \cdot (T_{\text{ini}} + L) + n - 1}$ number of data points used to construct $\mathcal{H}$ $\begin{cases} \text{For SISO:} & T_{\min} = 2L + n - 1 \\ \text{For MIMO:} & T_{\min} = (m+1) \cdot L + n - 1 \end{cases}$
- $\boxed{\text{col}_{\mathcal{H}} \geq T - L + 1}$ number of columns
- $\boxed{\text{row}_{\mathcal{H}} = L \cdot (m+p)}$ number of rows
- $\boxed{\underset{\max}{\text{rank}}(\mathcal{H}) = Lm + n}$ rank maximum (SISO: $L + n = K + 2n$)

- $Lm + n$ lin. ind. columns of $\mathcal{H}$ are chosen from data:

$$\mathcal{H}_{m \cdot L}(\mathbf{u}_{\text{data}}) = \begin{bmatrix} u_0 & u_1 & \cdots & u_{Lm+n-1} \\ u_1 & u_2 & \cdots & u_{Lm+n-2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{Lm-1} & u_{Lm} & \cdots & u_{2Lm+n-2} \end{bmatrix} \qquad \mathcal{H}_{p \cdot L}(\mathbf{y}_{\text{data}}) = \begin{bmatrix} y_0 & y_1 & \cdots & y_{Lm+n-1} \\ y_1 & y_2 & \cdots & y_{Lm+n-2} \\ \vdots & \vdots & \ddots & \vdots \\ y_{Lm-1} & y_{Lm} & \cdots & y_{2Lm+n-2} \end{bmatrix}$$

Considerations

- **Persistence of excitation** in the collected data ensures full rank and that all dynamics are represented.
- Adding data to $\mathcal{H}$ ensures enough data in case n is uncertain and all system modes have been exited.
- Ideally, $\mathcal{H}$ stops increasing after $mL + n$ (noiseless case).

4.3 DeePC

$$\min_{\mathbf{u},\mathbf{y},\mathbf{g},\boldsymbol{\sigma}} \sum_{k=0}^{K} \|\mathbf{y}_k - \mathbf{y}_{\text{ref}}\|_Q^2 + \|\mathbf{u}_k - \mathbf{u}_{\text{ref}}\|_R^2 + \lambda_g \|\mathbf{g}\|_1 + \lambda_\sigma \|\boldsymbol{\sigma}\|_1$$

$$\text{s.t.} \quad \underbrace{\begin{bmatrix} \mathcal{H}_{m \cdot L}(\mathbf{u}_{\text{data}}) \\ \mathcal{H}_{p \cdot L}(\mathbf{y}_{\text{data}}) \end{bmatrix}}_{\mathcal{H}} \mathbf{g} = \begin{bmatrix} \mathbf{u}_{\text{past}} \\ \mathbf{u}_{\text{fut}} \\ \mathbf{y}_{\text{past}} + \boldsymbol{\sigma} \\ \mathbf{y}_{\text{fut}} \end{bmatrix}$$

$$\mathbf{u}_k \in \mathcal{U}_k \quad \forall k$$
$$\mathbf{y}_k \in \mathcal{Y}_k \quad \forall k$$

Against noise and/or nonlinearities:

- Regularized/rank constraint: $\underset{\max}{\text{rank}}(\mathcal{H}) = Lm + n$:

Prevent overfitting to noise/mismatches

$$\begin{cases} \uparrow \lambda_g : & \text{fewer trajectories considered} \\ \text{too high } \lambda_g : & \text{over-regulation} \\ \downarrow \lambda_g : & \text{more trajectories considered} \\ \text{too low } \lambda_g : & \text{irrelevant/incorrect info used} \end{cases}$$

- Slack variable: slightly relax constraint:

noise/mismatches in initial conditions

$$\begin{cases} \uparrow \lambda_\sigma : & \text{more precise initial state selection} \\ \text{too high } \lambda_\sigma : & \text{poor tracking: cost dominated by } \lambda_\sigma \\ \downarrow \lambda_\sigma : & \text{more corrections possible} \\ \text{too low } \lambda_\sigma : & \text{poor estimation, poor tracking} \end{cases}$$

5 Markovian Decision Processes MDP - Optimal Control

Pros

- Discretized state space
- Nonlinear dynamics

Cons

- Model-based in the form of transition probabilities
- Memory to store π and V

Examples

- Traffic: number of vehicles per queue
- Stochastic processes: arrival of people, weather
- Fastest trajectory
- Games

- **Model** in form of probability transition: $\begin{cases} P: \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to [0,1) \\ P^u_{x,x'} = \mathbb{P}[x'|x,u] \end{cases}$ with set $\mathcal{X}$ of N states and set $\mathcal{U}$ of M inputs.
- **Markov property**: $P^u_{x,x'}$, $\forall x'$ only depends on x and u
- **Goal**: design a policy to select $u \in \mathcal{U}$ on current $x \in \mathcal{X}$: $\begin{cases} \text{deterministic} & \mu: \mathcal{X} \to \mathcal{U} \\ \text{stochastic} & \pi: \mathcal{X} \times \mathcal{U} \to [0,1) \quad \pi(x,u) = \mathbb{P}[u|x] \end{cases}$
- **Discounted Bellman Equation (contractive)**: Dynamic programming to be solved via backward recursion:

$$\begin{aligned} V_k^\pi(x) = V^\pi(x) &= \mathbb{E}\left[\sum_{i=k}^{K=\infty} \gamma^i c_i \middle| x_k = x\right] = \mathbb{E}\left[c_k + \sum_{i=k+1}^{K=\infty} \gamma^i c_i \middle| x_k = x\right] = \mathbb{E}\left[c_k | x_k = x\right] + \gamma \mathbb{E}\left[\sum_{i=k+1}^{K=\infty} c_i \middle| x_k = x\right] \\ &= \sum_u \pi_k(x,u) \cdot \mathbb{E}\left[c_k | x_k = x, u_k = u\right] + \gamma \sum_u \pi_k(x,u) \cdot \mathbb{E}\left[\sum_{i=k+1}^{K=\infty} c_i \middle| x_k = x, u_k = u\right] \\ &= \sum_u \pi_k(x,u) \left[\underbrace{\mathbb{E}\left[c_k | x_k = x, u_k = u\right]}_{C_x^u} + \gamma \sum_{x'} P^u_{x,x'} \underbrace{\mathbb{E}\left[\sum_{i=k+1}^{K=\infty} c_i \middle| x_{k+1} = x'\right]}_{V^\pi_{k+1}(x') = V^\pi(x')}\right] \end{aligned}$$

→ **Consistency equation**: Compute the value function V of a policy π based on model $P^u_{x,x'}$ and (immediate observable) cost C_x^u (with an estimate of future cost $V^\pi(x')$).
Infinite time: Stationary (no k for V, π), time invariant C_x^u and π.
With discount factor $\gamma \in [0,1)$ to ensure finite cost (Bernoulli termination probability).

- **Bellman Optimality Principle/Equation**: optimal sequential decision problems
Optimal value/cost:

$$\begin{aligned} V_k^\star(x) &= \min_{\pi_k} V_k^\pi(x) \\ &= \min_{\pi_k} \sum_u \pi_k(x,u) \left[C_x^u + \sum_{x'} P^u_{x,x'} V^\pi_{k+1}(x')\right] \end{aligned}$$

Inductively:

$$V_k^\star(x) = \min_{\pi_k} \sum_u \pi_k(x,u) \left[C_x^u + \sum_{x'} P^u_{x,x'} V^\star_{k+1}(x')\right]$$

→ Linear Program (LP) per x with base case V_K. At each step, a π is found.
→ Finite number of states and actions allows to tackle nonlinear stochastic systems with LP.

- For a **fixed policy** π, the MDP reduces to a **Markov Chain** with transition probabilities $P^\pi_{x,x'}$ and stationary distribution $\bar{d}(x)$:

$$P^\pi = \sum_u \mathbb{P}[x'|x,u]\pi(x,u) = \sum_u \pi(x,u) P^u_{x,x'}$$

Stationary point:

$$d_{k+1}^\top = d_k^\top P^\pi \quad \Longrightarrow \quad \boxed{\bar{d}^\top = \bar{d}^\top P^\pi}$$

Policy Iteration

Policy Evaluation: expensive: $\mathcal{O}(N^3 + N^2 M)$: inverse + multiply $N \times N$ with $N \times 1$, M times.

- Infinite time, **finite** MDP with N states $\Rightarrow$ system of N linear equations: $\boxed{V^\pi = C^\pi + \gamma P^\pi V^\pi}$

  with $\begin{cases} C^\pi(x) = \sum_u \pi(x,u) C_x^u & \text{expected cost } N \times 1 \\ P^\pi = \sum_u \pi(x,u) P_{x,x'}^u & \text{expected transition probabilities (row-stochastic) } N \times N \end{cases}$

  $V^\pi = (I - \gamma P^\pi)^{-1} C^\pi$ Always invertible (Perron-Frobenious see ATIC) and unique solution.

Greedy Policy Improvement: easy: min over M alternatives, N times.

- Always improve value ($V^{\pi'}(x) < V^\pi(x)$ unless $V^\pi = V^\star$) by deviating from current π for one step ν and then fall back to π:

$$\pi'(x,u) = \arg\min_\nu \sum_u \nu(x,u) \left[C_x^u + \gamma \sum_{x'} P_{x,x'}^u V^\pi(x') \right]$$

Convergence in finite number of steps because the number of actions and states is finite.

Algorithm 1 Policy Iteration Algorithm

```
1: Initialize at a policy guess $\pi \leftarrow \pi_0$
2: for each iteration do
3:     $V \leftarrow (I - \gamma P^\pi)^{-1} C^\pi$    ▷ Compute the value $V$ associated to the policy $\pi$
4:     $\pi(x,u) \leftarrow \arg\min_\nu \sum_u \nu(x,u) \left[C_x^u + \gamma \sum_{x'} P_{x,x'}^u V(x')\right]$    ▷ Greedy update of the policy to update $\pi$
5: end for
```

Value Iteration

Fixed point of the Bellman Optimality Equation: cheap: $\mathcal{O}(N^2 M)$

$$V^\star(x) = \min_\pi \sum_u \pi(x,u) \left[C_x^u + \gamma \sum_{x'} P_{x,x'}^u V^\star(x') \right]$$

Convergence asymptotically (after ∞ iterations) with rate γ to $V^\star$ (**contractive**).

Algorithm 2 Value Iteration Algorithm

```
1: Initialize at a value guess $V \leftarrow V_0$
2: for each iteration do
3:     $V(x) \leftarrow \min_\pi \sum_u \pi(x,u) \left[C_x^u + \gamma \sum_{x'} P_{x,x'}^u V(x')\right]$    ▷ Apply the Bellman iteration
4: end for
5: if convergence then
6:     $\pi(x,u) \leftarrow \arg\min_\nu \sum_u \nu(x,u) \left[C_x^u + \gamma \sum_{x'} P_{x,x'}^u V(x')\right]$    ▷ Extract the optimal policy $\pi^\star$ (greedy policy)
7: end if
```

6 Monte Carlo (Episodic) Learning

Pros

- Model-free (based on collected data) → use simulator (cheap) or controlled environment
- Parametrization via neural networks

Cons

- Averaging state-action pairs of full episode renders learning slow/inefficient (chess: 1 bad move and all perfect)
- No param.: exp. increasing matrix
- Param. reduces DoF & not solve Bellman equation

Examples

- Finance, managing portfolio
- Logistic inventory
- Drug development/simulation

Quality function Q returns the expect future cost of taking action u at state x and apply policy π afterwards.

- **Discounted Bellman Equation**:

$$V^\pi(x) = \sum_u \pi(x,u) \underbrace{\left[R_x^u + \gamma \sum_{x'} P_{x,x'}^u V^\pi(x') \right]}_{Q^\pi(x,u)} \qquad N \text{ states}$$

$$Q^\pi(x,u) = R_x^u + \gamma \sum_{x'} P_{x,x'}^u \underbrace{\sum_{u'} \pi(x',u') Q^\pi(x',u')}_{V^\pi(x')} = \mathcal{B}(Q) \qquad N \text{ states} \times M \text{ actions}$$

- **Bellman Optimality Principle/Equation**:

$$V^\star(x) = \min_u \left[R_x^u + \sum_{x'} P_{x,x'}^u V^\star(x') \right] = \min_u Q(x,u)$$

$$Q^\star(x,u) = R_x^u + \sum_{x'} P_{x,x'}^u \left(\min_{u'} Q^\star(x',u') \right)$$

Deterministic policy & infinite time
→ N nonlinear equations (min on finite set)

Swapped min with $\mathbb{E}$ (easier).
Both define implicitly optimal strategy.

Policy Iteration

Experimental Policy Evaluation: **No model**, from experiment/simulations (difficult and time consuming)

1. Collect a (long) episode: $(x_0, u_0, r_0),\ (x_1, u_1, r_1),\ \dots\ (x_T, u_T, r_T)$
2. Compute the **empirical cost** $g_0, g_1, \dots, g_T$
3. Interpret g_k as a realization of the return in the **state-action pair** x_k, u_k

$$Q^\pi(x_k, u_k) \approx g_k = \sum_{i=k}^{T} \gamma^{i-k} r_i$$

$$Q^\pi(x, u) = \frac{\sum_{t=1}^{T} 1[x_t = x, u_t = u]\, g_t}{\sum_{t=1}^{T} 1[x_t = x, u_t = u]} \qquad \text{in case of multiple visits}$$

Ergodicity assumption: all states visited.

Note: Policy evaluated (Q computed) only at the end of the episode, when agent sees final outcome.
Note: The estimation does not satisfy Bellman equation (unless with ∞ data).
Note: Markovianity of the physical system not exploited to make computation of Q more efficient $\rightarrow$ missing consistency.

Greedy Policy Improvement: **No model** information needed:

$$\pi'(x, u) = \arg\min_{\nu} \sum_u \nu(x, u) Q(x, u)$$

Note: Can find suboptimal solution: not learnt enough or optimal found and not exploring anymore.

Exploration and Exploitation:

ϵ-greedy Policy Improvement

$$\pi(x, u) \leftarrow \begin{cases} \arg\min_u Q(x, u) & \text{with probability } 1 - \epsilon \\ \text{Uniform}(\mathcal{U}) & \text{with probability } \epsilon \end{cases}$$

Large $\epsilon \rightarrow$ more exploration

Boltzmann Policy Improvement

$$\pi(x, u) \leftarrow \frac{e^{-\beta Q(x,u)}}{\sum_u e^{-\beta Q(x,u)}}$$

Small $\beta \rightarrow$ more exploration

- As learning progresses from episode to episode, $\epsilon \rightarrow 0$ or $\beta \rightarrow \infty$ (explore less, exploit more).
- If done "properly", then with probability 1
  - each state-action pair is visited **infinitely often**
  - the policy converges to a **greedy policy**

6.1 Parametrization of Q

- In theory: memorize Q in table $\rightarrow$ computationally infeasible (many state-action), intractable (continuous state-action).
- In practice: $\boxed{Q_\theta(x,u) = \phi^\top \theta = \Phi\theta}$ $\begin{cases} \Phi \in \mathbb{R}^{NM \times d} & \text{basis functions} \\ \theta \in \mathbb{R}^d \end{cases}$

Advantages

- Smaller memory (d instead of NM)
- Problem size is (apparently) independent from N $\rightarrow$ continuous state space
- A smart parametrization may allow to "guess" the Q function in state-action pairs that have not been observed (interpolation/extrapolation)
- Prior information on the problem may suggest a smart parametrization

Disadvantages

- Reduced degrees of freedom
- The solution to the Bellman equation for a given policy π may not belong to $\mathcal{Q} = \{Q^\theta, \theta \in \mathbb{R}^d\}$ (fixed point) because the solution is not of the same form.
- Possible consequences on
  - convergence of policy iterations
  - optimality of the limit
- Computing θ that best approximates the data collected from an episode may be computationally expensive

- **Q-Bellman Equation** $\mathcal{B}(Q)$:

$$\mathcal{B}(Q) = R_x^u + \gamma \underbrace{\sum_{x'} P_{x,x'}^u \sum_{u'} \pi(x',u')}_{T^\pi} Q^\pi(x',u') = R + \gamma T^\pi Q \qquad \begin{cases} Q \in \mathbb{R}^{NM} \\ R \in \mathbb{R}^{NM} \\ T^\pi \in \mathbb{R}^{NM \times NM} \end{cases}$$

- **Projected Bellman Equation**: **Model-free** best approximant:

$$Q_\theta = \underset{Q_\theta}{\text{argmin}} \|Q_\theta - \mathcal{B}(\mathcal{Q})\|_\rho^2 \qquad \text{with solution} \qquad \boxed{\Phi^\top \rho \Phi\theta = \gamma \Phi^\top \rho T^\pi \Phi\theta + \Phi^\top \rho R} \qquad \text{System of } d \text{ linear equations}$$

For every sample x_k, u_k, r_k of the episode and their successive sample x_{k+1}, u_{k+1}, we construct

- $\phi(x_k,u_k)\phi(x_k,u_k)^\top$
- $\phi(x_k,u_k)\phi(x_{k+1},u_{k+1})^\top$
- $\phi(x_k,u_k)r_k$

$$\phi(x_k,u_k) = \begin{bmatrix} \phi_1(x_k,u_k) \\ \vdots \\ \phi_d(x_k,u_k) \end{bmatrix}$$

The empirical average of these terms corresponds to the matrices above, where ρ encodes the frequency of each state-input pair in the episode.

7 (Online) Reinforcement Learning RL

Pros

- (almost) model-free
- Learn online during control $\rightarrow$ adaptive control
- Better realization of which action causes reward (Monte Carlo averages a full episode)

Cons

- Challenging learn and control (update π and Q at the same time)
- Few guarantees: suboptimal
- Parametrization: instabilities

Examples

- AI for games
- Autonomous vehicle
- Content recommendation (Netflix)

7.1 Temporal Difference TD

Policy evaluation method which updates Q based on Bellman's Principle. TD-learning iteratively adjusts $Q(x)$ to satisfy the equation $Q(x) = \mathbb{E}[r_k + \gamma Q(x')]$

- Assuming Bellman Equation satisfied with $\mathbb{E}[e_k] = 0$ and **TD error** $\boxed{e_k = r_k + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)}$
  Note: $\mathbb{E}$ needed because system is stochastic.
- Stochastic Approximation:

$$\boxed{q_{k+1} \leftarrow q_k + \alpha_k e(q_k)} \text{ converges to } q^\star \text{ of } \mathbb{E}[e(q)] = 0 \text{ if } \begin{cases} e(q) \text{ is bounded} \\ \mathbb{E}[e(q)] \text{ is non-decreasing in } q \text{ (and increasing at } q^\star) \\ \text{the sequence } \alpha_k \text{ satisfies } \sum_{k=0}^{\infty} \alpha_k = \infty,\ \sum_{k=0}^{\infty} \alpha_k^2 < \infty \end{cases}$$

Policy improvement step happens in a larger algorithm like **SARSA** or **Q-Learning**.

7.1.1 SARSA (State-Action-Reward-State-Action)

- Gradient-free,
- **On-policy**: it learns the value of the policy being followed, including exploration: uses u_{k+1}.

SARSA (conceptually similar to Policy Iteration)

Simultaneously two operations (instead of alternating as before):

1. Policy Evaluation via stochastic approximation of the **Bellman Equation** $(x_k, u_k, r_k, x_{k+1}, u_{k+1})$

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \big(\underbrace{r_k + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)}_{e_k} \big)$$
$$= (1 - \alpha_k) Q(x_k, u_k) + \alpha_k \big(r_k + \gamma Q(x_{k+1}, u_{k+1}) \big)$$

2. Policy Improvement: improves the followed policy (could be ϵ-greedy) naturally.

As learning progresses $\epsilon \rightarrow 0$ or $\beta \rightarrow \infty$ (explore less, exploit more).

7.1.2 Q-Learning

- Gradient-free,
- **Off-policy**: it learns the value of the optimal policy independently of the agent's actions (behavior policy): no u_{k+1}.

Bellman Optimality Principle

$$Q^\star(x, u) = R_x^u + \gamma \sum_{x'} P_{x,x'}^u \left(\min_{u'} Q^\star(x', u') \right)$$

Individual realization (in $\mathbb{E}$ equal to Bellman Optimality Principle)

$$Q^\star(x_k, u_k) = r_k + \gamma \min_{u_{k+1}} Q^\star(x_{k+1}, u_{k+1})$$

- During Learning: As the Q-values are updated after each action, the greedy policy (optimal policy) is improved incrementally. You don't have to explicitly apply policy improvement after every Q-value update because Q-learning inherently optimizes the Q-values with respect to the best future actions.
- After Learning: Once the Q-learning process has converged (i.e., the Q-values have stabilized), you can extract the final optimal policy by simply selecting the action that minimizes the Q-value in each state.

Q-Learning (conceptually similar to Value Iteration)

Free two operations (instead of alternating as before):

1. Policy Evaluation via stochastic approximation of the **Bellman Optimality Equation** (x_k, u_k, r_k, x_{k+1})

$$\begin{aligned} Q(x_k, u_k) &\leftarrow Q(x_k, u_k) + \alpha_k \Big(r_k + \gamma \min_u Q(x_{k+1}, u) - Q(x_k, u_k) \Big) \\ &= (1-\alpha_k) Q(x_k, u_k) + \alpha_k \big(r_k + \gamma \min_u Q(x_{k+1}, u) \big) \end{aligned}$$

2. Policy Improvement: learns the optimal (greedy) policy independent of the policy being followed during exploration (implicitly).

- Forward dynamic programming

Note: Q-values are updated to reflect the best possible actions (conceptually similar to value iteration).

7.1.3 Parametrized Q-Learning as Stochastic Gradient Descent

As Q is huge (scalability issue) → approximate it → Bellman Optimality Principle (most likely) does not have a solution.

Individual realization:

$$Q^\star_\theta = \phi^\top(x_k, u_k)\theta^\star = \underbrace{r_k + \gamma \min_{u_{k+1}} \phi^\top(x_{k+1}, u_{k+1})\theta_k}_{Q^+}$$

Loss function:

$$\min_\theta \underbrace{\frac{1}{2}\left(\phi^\top(x,u)\theta - Q^+\right)^2}_{L(\theta)}$$

A simple iteration that converges to this minimum is the **gradient descent**:

$$\begin{aligned} \theta_{i+1} &= \theta_i - \alpha_i \nabla L(\theta_i) \\ &= \theta_i - \alpha_i \left(\phi^\top(x_k, u_k)\theta_i - \Big(\underbrace{r_k + \gamma \min_u \phi^\top(x_{k+1}, u)\theta_i}_{Q^+} \Big) \right) \phi(x_k, u_k) \end{aligned}$$

7.2 Policy Gradient

Pros

- Model-free: no V or Q to learn optimal π

Cons

- Few guarantees: suboptimal
- Parametrization: instability

Examples

- Natural language processing
- Real time game strategies

Both Monte Carlo and Temporal Difference methods work on state representability (i.e. not completely model free). Policy Gradient methods learn the optimal policy π without learning V or Q.

- Consider a **trajectory** τ of the system (assuming x_0 fixed over many episodes): $\tau = (x_0, u_0, x_1, u_1, \dots, x_T, u_T)$
- Associated reward/cost: $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$
- Stochastic policy parametrized in θ: $\pi_\theta(x, u)$
- Probability of trajectory τ happening when using policy π_θ: $\mathbb{P}_\theta(\tau) = \prod_{t=0}^{T} \underbrace{\mathbb{P}(x_{t+1} \mid x_t, u_t)}_{\text{transition probabilities}} \underbrace{\pi_\theta(u_t, x_t)}_{\text{policy}}$

Goal: minimize the expected cost: $J(\theta) := \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \sum_\tau \mathbb{P}_\theta(\tau) R(\tau)$

$$\nabla J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla \log \mathbb{P}_\theta(\tau) R(\tau)\right]$$

where

$$\nabla \log \mathbb{P}_\theta(\tau) = \sum_{t=0}^{T} \nabla \log \pi_\theta(u_t, x_t)$$

only depends on the parametrized policy! $\rightarrow$ gradient descent using only rewards and policy.

Algorithm 3 Policy Gradient Algorithm

```
1: Generate $\tau$ via $\pi_\theta$
2: Compute $R(\tau)$
3: Update $\theta \leftarrow \theta - \alpha R(\tau) \sum_{t=0}^{T} \nabla \log \pi_\theta(u_t, x_t)$
```

Example: SARSA vs. Q-Learning

Environment: A 3x3 grid where the agent starts at position (1, 1) and the goal is to reach position (3, 3).
Actions: *Up, Down, Left, Right*
Rewards:

- Reaching the goal (3, 3): +10
- Every other step: -1 (to encourage reaching the goal quickly).

Discount Factor: $\gamma = 0.9$
Learning Rate: $\alpha = 0.1$

SARSA Example (On-Policy)

1. **Initialize Q-values**: Start with all Q-values $Q(x, u)$ initialized to 0.
2. **Episode Start**: Assume the agent is at position (1, 1).

- **State**: $x_1 = (1, 1)$
- **Action**: Use ϵ-greedy to select an action. Suppose $\epsilon = 0.1$, meaning the agent has a 90% chance of choosing the action with the highest Q-value and a 10% chance of exploring.
- Initially, all Q-values are equal, so assume the agent explores and selects $u_1 = \text{Right}$.

3. **Step 1**:

- **Take Action**: The agent moves to position $x_2 = (1, 2)$.
- **Reward**: The agent receives a reward $r_1 = -1$ (for moving without reaching the goal).
- **Next Action**: Again, use ϵ-greedy to select the next action. Suppose the agent selects $u_2 = \text{Down}$.
- **Update Q-value**: Update $Q(x_1, u_1)$ using the SARSA update rule:

$$Q(x_1, u_1) \leftarrow Q(x_1, u_1) + \alpha \left[r_1 + \gamma Q(x_2, u_2) - Q(x_1, u_1)\right]$$

$$Q((1, 1), \text{Right}) \leftarrow 0 + 0.1 \left[-1 + 0.9 \cdot Q((1, 2), \text{Down}) - 0\right]$$

  Since $Q((1, 2), \text{Down}) = 0$ initially, the update is:

$$Q((1, 1), \text{Right}) \leftarrow -0.1$$

4. **Step 2**:

- **State**: $x_2 = (1, 2)$
- **Action**: The agent takes the action $u_2 = \text{Down}$.
- **Next State**: The agent moves to position $x_3 = (2, 2)$.
- **Next Action**: Select $u_3 = \text{Right}$ using ϵ-greedy.
- **Update Q-value**: Update $Q((1, 2), \text{Down})$ using the SARSA update rule.

Q-Learning Example (Off-Policy)

1. **Initialize Q-values**: Start with all Q-values $Q(x, u)$ initialized to 0.
2. **Episode Start**: Assume the agent is at position (1, 1).

- **State**: $x_1 = (1, 1)$
- **Action**: Use ϵ-greedy to select an action. Suppose the agent selects $u_1 = \text{Right}$.

3. **Step 1**:

- **Take Action**: The agent moves to position $x_2 = (1, 2)$.
- **Reward**: The agent receives a reward $r_1 = -1$.
- **Next Action (for exploration)**: The agent might explore and choose $u_2 = \text{Down}$ using ϵ-greedy.

- **Update Q-value**: Instead of using $Q(x_2, u_2)$ (as in SARSA), Q-learning updates $Q(x_1, u_1)$ using the maximum Q-value of the next state:

  $$Q(x_1, u_1) \leftarrow Q(x_1, u_1) + \alpha \left[r_1 + \gamma \max_u Q(x_2, u) - Q(x_1, u_1)\right]$$

  $$Q((1,1), \text{Right}) \leftarrow 0 + 0.1 \left[-1 + 0.9 \cdot \max_u Q((1,2), u) - 0\right]$$

  Since all Q-values are initially 0, the update is:

  $$Q((1,1), \text{Right}) \leftarrow -0.1$$

4. **Step 2**:

- **State**: $x_2 = (1, 2)$
- **Action**: The agent takes the action $u_2 = \text{Down}$.
- **Next State**: The agent moves to position $x_3 = (2, 2)$.
- **Next Action (for update)**: Regardless of the action actually taken, Q-learning uses the action that maximizes the Q-value in state x_3 to update $Q((1, 2), \text{Down})$.