COSC-6590/GSCS-6390

Games: Theory and Applications

Lecture 15 - One-Player Dynamic Games

Luis Rodolfo Garcia Carrillo

School of Engineering and Computing Sciences
Texas A&M University - Corpus Christi, USA

Table of contents

One-Player Discrete-Time Games

One-Player Discrete-Time Games

Solution methods for one-player (discrete-time) dynamic games, which are simple optimizations

This corresponds to dynamics of the form

$$\underbrace{x_{k+1}}_{\substack{\text{entry node at} \\ \text{stage } k+1}} = \underbrace{f_k}_{\substack{\text{“dynamics”} \\ \text{at stage } k}} \Bigg(\underbrace{x_k}_{\substack{\text{state at} \\ \text{stage } k}} , \underbrace{u_k}_{\substack{P_1\text{'s action} \\ \text{at stage } k}} \Bigg) \quad \forall k \in \{1, 2, \ldots, K\}$$

starting at some initial state x_1 in the state space $\mathcal{X}$.

At each time k, the action u_k is required to belong to a given action space $\mathcal{U}_k$.

One-Player Discrete-Time Games

Assume finite horizon ($K < \infty$) stage-additive costs of the form

$$J := \sum_{k=1}^{K} g_k(x_k, u_k)$$

that the (only) player wants to minimize using either:

Open-Loop (OL) policy

$$u_k = \gamma_k^{\mathrm{OL}}(x_1), \qquad \forall k \in \{1, 2, \ldots, K\}$$

State-Feedback (FB) policy

$$u_k = \gamma_k^{\mathrm{FB}}(x_k), \qquad \forall k \in \{1, 2, \ldots, K\}$$

Discrete-Time Cost-To-Go

Discrete-Time Cost-To-Go

Suppose the player is at some **state** x at **stage** ℓ

- x would perhaps **not** be the **optimal** place to be at ℓ
- still, the player wants to **estimate the cost**, if playing optimally from this point on, so as **to minimize costs** incurred in remaining stages.

Cost-to-Go from state $x \in \mathcal{X}$ at time $\ell \in \{1, 2, \dots, K\}$

$$V_\ell(x) := \inf_{u_\ell \in \mathcal{U}_\ell, u_{\ell+1} \in \mathcal{U}_{\ell+1}, \dots, u_k \in \mathcal{U}_K} \sum_{k=\ell}^{K} g_k(x_k, u_k), \quad \forall x \in \mathcal{X}$$

with the sequence $\{x_k \in \mathcal{X} : k = \ell, \ell + 1, \dots, K\}$ starting at $x_\ell = x$ and satisfying the dynamics

$$x_{k+1} = f_k(x_k, u_k) \qquad \forall k \in \{\ell, \ell + 1, \dots, K - 1\}$$

Discrete-Time Cost-To-Go

Note: The **cost-to-go** is a function of x and ℓ.
Often called the **value function** of the game/optimization.

Computing the cost-to-go $V_1(x_1)$ from the initial state x_1 at the first stage $\ell = 1$ essentially amounts to minimizing the cost

$$J := \sum_{k=1}^{K} g_k(x_k, u_k)$$

for the dynamics

$$\underbrace{x_{k+1}}_{\substack{\text{entry node at} \\ \text{stage } k+1}} = \underbrace{f_k}_{\substack{\text{“dynamics”} \\ \text{at stage } k}} \Big(\underbrace{x_k}_{\substack{\text{state at} \\ \text{stage } k}} , \underbrace{u_k}_{\substack{P_1\text{'s action} \\ \text{at stage } k}} \Big) \quad \forall k \in \{1, 2, \ldots, K\}$$

This leads to two important conclusions.

Discrete-Time Cost-To-Go

Conclusion 1. Regardless of the information structure considered (OL, FB, other), it is not possible to obtain a cost

$$J := \sum_{k=1}^{K} g_k(x_k, u_k)$$

lower than $V_1(x_1)$.

This is because in the minimization

$$V_\ell(x) := \inf_{u_\ell \in \mathcal{U}_\ell, u_{\ell+1} \in \mathcal{U}_{\ell+1}, \ldots, u_k \in \mathcal{U}_K} \sum_{k=\ell}^{K} g_k(x_k, u_k), \quad \forall x \in \mathcal{X}$$

we place no constraints on what information may or may not be available to compute the optimal u_k.

- $V_1(x_1)$: lower bound on the smallest value achieved for J.

Discrete-Time Cost-To-Go

Conclusion 2. If the infimum in the minimization

$$V_\ell(x) := \inf_{u_\ell \in \mathcal{U}_\ell, u_{\ell+1} \in \mathcal{U}_{\ell+1}, \ldots, u_k \in \mathcal{U}_K} \sum_{k=\ell}^{K} g_k(x_k, u_k), \quad \forall x \in \mathcal{X}$$

is achieved for a specific sequence

$$u_1^* \in \mathcal{U}_1, u_2^* \in \mathcal{U}_2, \ldots, u_K \in \mathcal{U}_K$$

computed before the game starts just with knowledge of x_1, then this sequence of actions provides an **optimal OL policy**

$$\gamma_1^{\mathrm{OL}}(x_1) := u_1^*, \quad \gamma_2^{\mathrm{OL}}(x_1) := u_2^*, \ldots, \quad \gamma_K^{\mathrm{OL}}(x_1) := u_K^*,$$

Here, $V_1(x_1)$ is the smallest value that can be achieved for J.

This would not be the case, e.g., if there were stochastic events

Discrete-Time Dynamic Programming

Discrete-Time Dynamic Programming

DP is a computationally efficient recursive technique

- **useful to compute the cost-to-go**

For the last stage K, the cost-to-go $V_K(x)$ is the minimum of

$$g_K(x_K, u_K)$$

over the possible actions u_K, for a game that starts with $x_K = x$, and therefore

$$V_K(x) = \inf_{u_K \in \mathcal{U}_K} g_K(x, u_K), \qquad \forall x \in \mathcal{X}$$

Note: When $g_K(\cdot)$ is continuously differentiable, the optimization can be done using calculus by solving

$$\frac{dg_K(x_K, u_K)}{du_K} = 0$$

Discrete-Time Dynamic Programming

For each state x, we compute $V_K(x)$ by solving a single parameter optimization over the set $\mathcal{U}_K$.

For the previous stages $\ell < K$, we have that

$$
\begin{aligned}
V_\ell(x) :=& \inf_{u_\ell \in \mathcal{U}_\ell, \ldots, u_K \in \mathcal{U}_K} \sum_{k=\ell}^{K} g_k(x_k, u_k) \\
=& \inf_{u_\ell \in \mathcal{U}_\ell, \ldots, u_K \in \mathcal{U}_K} \left(\underbrace{g_\ell(x, u_\ell)}_{\substack{\text{independent of} \\ u_{\ell+1}, \ldots, u_K}} + \underbrace{\sum_{k=\ell+1}^{K} g_k(x_k, u_k)}_{\substack{\text{dependent on all} \\ u_\ell, \ldots, u_K}} \right) \\
=& \inf_{u_\ell \in \mathcal{U}_\ell} \left(g_\ell(x, u_\ell) + \inf_{u_{\ell+1} \in \mathcal{U}_{\ell+1}, \ldots, u_K \in \mathcal{U}_K} \sum_{k=\ell+1}^{K} g_k(x_k, u_k) \right)
\end{aligned}
$$

Discrete-Time Dynamic Programming

Where we used these facts:

- first equality: we must set $x_\ell = x$ to compute $V_\ell(x)$
- second equality: $g_\ell(x, u_\ell)$ does not depend on $u_{\ell+1}, \dots, u_K$

However

$$\inf_{u_{\ell+1} \in \mathcal{U}_{\ell+1}, \dots, u_K \in \mathcal{U}_K} \sum_{k=\ell+1}^{K} g_k(x_k, u_k)$$

is the minimum cost for a game starting at stage $\ell + 1$ with state

$$x_{\ell+1} = f_\ell(x, u_\ell)$$

which is precisely the cost-to-go $V_{\ell+1}(f_\ell(x, u_\ell))$.

Discrete-Time Dynamic Programming

We therefore conclude

$$V_\ell(x) = \inf_{u_\ell \in \mathcal{U}_\ell} \Big(g_\ell(x, u_\ell) + V_{\ell+1}(f_\ell(x, u_\ell)) \Big), \quad \forall x \in \mathcal{X}, \quad \ell \in \{1, 2, \dots, K-1\}$$

Note: If we know the function $V_{\ell+1}(\cdot)$, we can compute each $V_\ell(x)$ by solving a single-parameter optimization over set $\mathcal{U}_\ell$. This optimization produces the optimal action u_ℓ^* to be used when the state is at x_ℓ.

It is convenient to define $V_{K+1}(x) = 0, \quad \forall x \in \mathcal{X}$

Allowing us to re-write $V_K(x)$ and $V_\ell(x)$ using

$$V_\ell(x) = \inf_{u_\ell \in \mathcal{U}_\ell} \Big(g_\ell(x, u_\ell) + V_{\ell+1}(f_\ell(x, u_\ell)) \Big), \quad \forall x \in \mathcal{X}$$

now valid $\forall \ell \in \{1, 2, \dots, K\}$

Discrete-Time Dynamic Programming

For the case of $\ell = 1$ and $x = x_1$ and when the infima in $V_\ell(x)$ are actually minima, the points at which these infima are achieved can be used to construct an open-loop policy.

Specifically, we can obtain:

u_1^* from $V_\ell(x)$ with $\ell = 1$ and $x = x_1$,

- leading to $x_2^* = f_1(x_1, u_1^*)$;

u_2^* from $V_\ell(x)$ with $\ell = 2$ and $x = x_2^*$,

- leading to $x_3^* = f_2(x_2^*, u_2^*)$;

u_3^* from $V_\ell(x)$ with $\ell = 3$ and $x = x_3^*$,

- leading to $x_4^* = f_3(x_3^*, u_3^*)$;

etc. . .

Open-Loop Optimization

Procedure to compute the **OL policy** γ^{OL} that minimizes

$$J := \sum_{k=1}^{K} g_k(x_k, u_k)$$

for the dynamics

$$\underbrace{x_{k+1}}_{\substack{\text{entry node at}\\ \text{stage } k+1}} = \underbrace{f_k}_{\substack{\text{"dynamics"}\\ \text{at stage } k}} \Big(\underbrace{x_k}_{\substack{\text{state at}\\ \text{stage } k}} , \underbrace{u_k}_{\substack{P_1\text{'s action}\\ \text{at stage } k}} \Big) \quad \forall k \in \{1, 2, \ldots, K\}$$

Step 1: Compute the cost-to-go using **backward iteration** starting from $\ell = K$, proceeding backward in time until $\ell = 1$

$$V_{K+1}(x) = 0, \quad V_\ell(x) = \inf_{u_\ell \in \mathcal{U}_\ell} \Big(g_\ell(x, u_\ell) + V_{\ell+1}(f_\ell(x, u_\ell)) \Big), \quad \forall x \in \mathcal{X}$$

Note: To do the backwards iteration, compute each $V_\ell(x)$ for every possible value of the state x at stage ℓ.

Open-Loop Optimization

Step 2: Compute the sequence of actions

$$u_1^* \in \mathcal{U}_1, u_2^* \in \mathcal{U}_2, \ldots, u_k^* \in \mathcal{U}_K$$

that minimize $V_1(x_1)$ using a **forward iteration**, starting from $k = 1$ and proceeding forward in time until $k = K$:

$$x_1^* = x_1, u_k^* = \underset{u_k \in \mathcal{U}_K}{\arg\min} \underbrace{\Big(g_k(x_k^*, u_k) + V_{k+1}(f_k(x_k^*, u_k))\Big)}_{\text{computed using the precomputed states } x_k^*}, x_{k+1}^* = f_k(x_k^*, u_k^*)$$

Assumption: infimum of $g_\ell(x_\ell^*, u_\ell) + V_{\ell+1}(f_\ell(x_\ell^*, u_\ell))$ is achieved at some point $u_k \in \mathcal{U}_K$.

- if this is not the case, then this procedure fails.

When the infimum is achieved at multiple points, any one can be used in the equation.

Open-Loop Optimization

Step 3: Finally, the optimal OL policy γ^{OL} is given by

$$\gamma_1^{\mathrm{OL}}(x_1) := u_1^*, \quad \gamma_2^{\mathrm{OL}}(x_1) := u_2^*, \quad \ldots, \quad \gamma_K^{\mathrm{OL}}(x_1) := u_K^*,$$

Observation: All the x_k^* and u_k^* in **Step 2** are precomputed and depend solely on the initial state x_1.

Thus, γ^{OL} is indeed an OL policy.

State-Feedback Optimization

Suppose we use the optimal OL policy γ^{OL} defined in **Step 3**, which selects the actions

$$u_k = \gamma_k^{\text{OL}}(x_1) := u_k^*, \quad \forall k \in \{1, 2, \ldots, K\}$$

In this case, the precomputed states x_k^* defined in **Step 2** match precisely the states x_k that would be measured during the game.

Therefore, we would get the same minimum value $V_1(x_1)$ for the cost J, if we were using a state-FB policy γ^{FB} defined by

$$\gamma_k^{\text{FB}}(x_k) := \arg\min_{u_k \in \mathcal{U}_K} \underbrace{\Big(g_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k))\Big)}_{\text{computed using the measured state } x_K}, \quad \forall k \in \{1, 2, \ldots, K\}$$

State-Feedback Optimization

When all the $g_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k))$ have a minimum for some $u_k \in \mathcal{U}_K$, this state-FB policy γ^{FB} can do as well as the optimal OL policy γ^{OL}.

Since it is not possible to obtain a value for the cost J lower than $V_1(x_1)$, we conclude that $\gamma_k^{\mathrm{FB}}(x_k)$ is an optimal FB policy.

Notation 5 (Time-consistent policy).

A state-FB policy $\gamma_k^{\mathrm{FB}}(x_k)$ that minimizes the cost-to-go from current state x_k at time k is said to be **time consistent**.

There may be policies $\bar{\gamma}^{\mathrm{FB}}$ that still achieve a cost as low as $V_1(x_1)$, but are not time consistent because $\bar{\gamma}_k^{\mathrm{FB}}(x_k)$ may not achieve the minimum in $\gamma_k^{\mathrm{FB}}(x_k)$ for every state x_k.

State-Feedback Optimization

Why? it is irrelevant for a policy to achieve the minimum in $\gamma_k^{\mathrm{FB}}(x_k)$ for states x_k never reached through an optimal path.

Time-consistent policies are robust

If due to an unexpected event the state at some time k is taken to a point other than

$$x_{k+1} \neq f_k(x_k, u_k)$$

then a time-consistent policy is still optimal in minimizing the cost-to-go from the stage $k+1$ forward.

OL policies are not robust because they rely on precomputed states and cannot react to unexpected events.

State-Feedback Optimization

Theorem 15.1. Consider the sequence of functions $V_1(x), V_2(x), \ldots, V_{K+1}(x)$ uniquely defined by

$$V_k(x) = \begin{cases} 0 & k = K+1 \\ \inf_{u_k \in \mathcal{U}_k} \Big(g_k(x, u_k) + V_{k+1}(f_k(x, u_\ell)) \Big) & k \in \{1, 2, \ldots, K\}, \end{cases}$$

$\forall x \in \mathcal{X}$. Then $V_k(x)$ is equal to the cost-to-go, and if the infimum is always achieved at some point in $\mathcal{U}_k$, we have that:

1. For any initial state x_1, an optimal OL policy γ^{OL} is

$$\gamma^{\mathrm{OL}}(x_1) := u_k^*, \quad \forall k \in \{1, 2, \ldots, K\},$$

with u_k^* obtained from solving

$$x_1^* = x_1, u_k^* = \arg\min_{u_k \in \mathcal{U}_K} \underbrace{\Big(g_k(x_k^*, u_k) + V_{k+1}(f_k(x_k^*, u_k)) \Big)}_{\text{computed using the precomputed states } x_k^*}, x_{k+1}^* = f_k(x_k^*, u_k^*)$$

State-Feedback Optimization

Note. In an OL setting, both x_k^* and u_k^*, $\forall k \in \{1, 2, \ldots, K\}$ are precomputed before the game starts.

2. An optimal (time-consistent) state-FB policy γ^{FB} is

$$\gamma_k^{\mathrm{FB}}(x_k) := \underset{u_k \in \mathcal{U}_K}{\arg\min} \underbrace{\Big(g_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k))\Big)}_{\text{computed using the measured state } x_K}, \quad \forall k \in \{1, 2, \ldots, K\}$$

Either of the above optimal policies leads to an optimal cost equal to $V_1(x_1)$.

State-Feedback Optimization

Proof of Theorem 15.1.

Let u_k^* and x_k^*, $\forall k \in \{1, 2, \ldots, K\}$ be a trajectory arising from the OL or the state-FB policies.

Let $\bar{u}_k$ and $\bar{x}_k$, $\forall k \in \{1, 2, \ldots, K\}$ be another (arbitrary) trajectory.

To prove optimality, show that the latter trajectory cannot lead to a cost lower than the former.

Since $V_k(x)$ satisfies the conditions in Theorem 15.1, and u_k^* achieves the infimum in $V_k(x)$, for every $k \in \{1, 2, \ldots, K\}$

$$
\begin{aligned}
V_k(x_k^*) &= \inf_{u_k \in \mathcal{U}_k} \Big((g_k(x_k^*, u_k) + V_{k+1}(f_k(x_k^*, u_k)) \Big) \\
&= g_k(x_k^*, u_k^*) + V_{k+1}(f_k(x_k^*, u_k^*)
\end{aligned}
$$

State-Feedback Optimization

Since $\bar{u}_k$ does not necessarily achieve the infimum, we have

$$\begin{aligned} V_k(\bar{x}_k) &= \inf_{u_k \in \mathcal{U}_k} \Big((g_k(\bar{x}_k, u_k) + V_{k+1}(f_k(\bar{x}_k, u_k)) \Big) \\ &\leq g_k(\bar{x}_k, \bar{u}_k) + V_{k+1}(f_k(\bar{x}_k, \bar{u}_k)) \end{aligned}$$

Summing both sides of $V_k(x_k^*)$ from $k = 1$ to $k = K$, we conclude

$$\sum_{k=1}^{K} V_k(x_k^*) = \sum_{k=1}^{K} g_k(x_k^*, u_k^*) + \sum_{k=1}^{K} V_{k+1}\Big(\underbrace{f_k(x_k^*, u_k^*)}_{x_{k+1}^*} \Big)$$

$$\Leftrightarrow \sum_{k=1}^{K} V_k(x_k^*) - \sum_{k=1}^{K} V_{k+1}(x_{k+1}^*) = \sum_{k=1}^{K} g_k(x_k^*, u_k^*)$$

State-Feedback Optimization

Since

$$\sum_{k=1}^{K} V_k(x_k^*) - \sum_{k=1}^{K} V_{k+1}(x_{k+1}^*) = V_1(x_1) - V_{K+1}(x_{K+1}^*) = V_1(x_1)$$

We conclude that

$$V_1(x_1) = \sum_{k=1}^{K} g_k(x_k^*, u_k^*)$$

Now summing both sides of $V_k(\bar{x}_k)$ from $k = 1$ to $k = K$

$$\sum_{k=1}^{K} V_k(\bar{x}_k) \leq \sum_{k=1}^{K} g_k(\bar{x}_k, \bar{u}_k) + \sum_{k=1}^{K} V_{k+1}\Big(\underbrace{f_k(\bar{x}_k, \bar{u}_k)}_{\bar{x}_{k+1}}\Big)$$

$$\Leftrightarrow \sum_{k=1}^{K} V_k(\bar{x}_k) - \sum_{k=1}^{K} V_{k+1}(\bar{x}_{k+1}) \leq \sum_{k=1}^{K} g_k(\bar{x}_k, \bar{u}_k)$$

State-Feedback Optimization

We conclude that

$$V_1(x_1) \leq \sum_{k=1}^{K} g_k(\bar{x}_k, \bar{u}_k)$$

from which we obtain

$$V_1(x_1) = \sum_{k=1}^{K} g_k(x_k^*, u_k^*) \leq \sum_{k=1}^{K} g_k(\bar{x}_k, \bar{u}_k)$$

Two conclusions can be drawn from this equation

1. The signal $\bar{u}_k$ does not lead to a cost that is smaller than that of u_k^*.

2. $V_1(x_1)$ is equal to the optimal cost obtained with u_k^*.

State-Feedback Optimization

If we had carried out the above proof on an interval $\{\ell, \ell+1, \ldots, K\}$ with initial state $x_\ell = x$, we would have concluded: $V_\ell(x)$ is the (optimal) value of the cost-to-go from state x at time ℓ.

Note 13. In the proof of **Theorem 15.1** we showed that it is not possible to achieve a cost lower than $V_1(x_1)$, regardless of the information structure.

This is because the signal $\bar{u}_k$ considered could have been generated by a policy using any information structure and we showed that $\bar{u}_k$ cannot lead to a cost smaller than $V_1(x_1)$.

Computational Complexity

Computational Complexity

For large state-spaces $\mathcal{X}$, the computational effort needed to compute the cost-to-go at all stages can be very large.

Question: **Is it worth using dynamic programming, instead of doing an exhaustive search?**

- to decide which option is best, estimate the computation involved in exploring each option.

Assumption: finite state-spaces and finite action spaces.

Exhaustive Search. Suppose a game has K stages. At the stage ℓ the number of actions available to the player is $|\mathcal{U}_\ell|$.

An exhaustive search over all possible selections of actions requires comparing the costs associated with as many options as

$$|\mathcal{U}_1| \times |\mathcal{U}_2| \times \cdots \times |\mathcal{U}_K|$$

Computational Complexity

Dynamic Programming. At a particular stage ℓ and for a specific value of the state x, computing the cost-to-go $V_\ell(x)$ requires comparing all the actions available, which requires $|\mathcal{U}_\ell|$ comparisons.

This has to be done for every state x and for every stage $\ell \in \{1, 2, \ldots, K\}$. The total number of comparisons is equal to

$$|\mathcal{U}_1| \times |\mathcal{X}_1| + |\mathcal{U}_2| \times |\mathcal{X}_2| + \cdots + |\mathcal{U}_K| \times |\mathcal{X}_K|$$

where $|\mathcal{X}_\ell|$ denotes the number of possible states at the stage ℓ.

Dynamic Programming can result in significant savings provided that the size of the state space is small when compared to **Exhaustive Search**.

Computational Complexity

Example 15.1 (Tic-Tac-Toe).

Consider a (silly) version of the Tic-Tac-Toe game in which the same player places all the marks.

An **Exhaustive Search** among all possible ways to play would have to consider

- 9 possible ways to place the first $\times$
- 8 possible ways to place the subsequent $\circ$
- 7 possible ways to to place the first $\times$
- etc.,

leading to a total of

$$9! = 9 \times 8 \times \cdot \times 2 \times 1 = 362880$$

distinct options that must be compared.

Computational Complexity

For **Dynamic Programming**, the total number of comparisons needed turns out to be about 19 times smaller

Stage	Number of $\times$'s	Number of $\circ$'s	$\lvert\mathscr{X}_\ell\rvert$	$\lvert\mathscr{U}_\ell\rvert$	$\lvert\mathscr{X}_\ell\rvert \times \lvert\mathscr{U}_\ell\rvert$
1	0	0	1	9	9
2	1	0	9	8	72
3	1	1	$9 \times 8 = 72$	7	504
4	2	1	$\binom{9}{2} \times 7 = 252$	6	1512
5	2	2	$\binom{9}{2} \times \binom{7}{2} = 756$	5	3780
6	3	2	$\binom{9}{3} \times \binom{6}{2} = 1260$	4	5040
7	3	3	$\binom{9}{3} \times \binom{6}{3} = 1680$	3	5040
8	4	3	$\binom{9}{4} \times \binom{5}{3} = 1260$	2	2520
9	4	4	$\binom{9}{4} \times \binom{5}{4} = 630$	1	630
10	5	4	$\binom{9}{5} = 126$	0	0
			Total number of comparisons needed		19107

In larger games, the difference between the two approaches is even more spectacular. **This happens because many different sequences of actions collapse to the same state.**

Solving Finite One-Player Games with MATLAB

Solving Finite One-Player Games with MATLAB

For games with finite state spaces and finite action spaces, the backwards iteration

$$V_{K+1}(x) = 0, \quad V_\ell(x) \inf_{u_\ell \in \mathcal{U}_\ell} \Big(g_\ell(x, u_\ell) + V_{\ell+1}(f_\ell(x, u_\ell)) \Big), \quad \forall x \in \mathcal{X}$$

can be implemented very efficiently in **MATLAB®**.

Enumerate all states so that the state-space can be viewed as

$$\mathcal{X} := \{1, 2, \ldots, n_{\mathcal{X}}\}$$

Enumerate all actions so that the action space can be viewed as

$$\mathcal{U} := \{1, 2, \ldots, n_{\mathcal{U}}\}$$

Assume that all states can occur at every stage and that all actions are also available at every stage.

Solving Finite One-Player Games with MATLAB

Each function $f_k(x,u)$ and $g_k(x,u)$ that define the game dynamics and the stage-cost, can be represented by an $n_{\mathcal{X}} \times n_{\mathcal{U}}$ matrix with one row per state, one column per action.

Each cost-to-go $V_k(x)$ can be represented by an $n_{\mathcal{X}} \times 1$ column vector with one row per state.

Construct the following variables within **MATLAB®**

`F` : a cell-array with K elements, each equal to an $n_{\mathcal{X}} \times n_{\mathcal{U}}$ matrix so that `F{k}` represents the game dynamics function $f_k(x,u)$, $\forall x \in \mathcal{X}$, $u \in \mathcal{U}$, $k \in \{1,2,\ldots,K\}$.

Specifically, the entry `F{k}(i,j)` of the matrix `F{k}` is the state $f_k(i,j)$.

Solving Finite One-Player Games with MATLAB

`G` : a cell-array with K elements, each equal to an $n_{\mathcal{X}} \times n_{\mathcal{U}}$ matrix so that `G{k}` represents the stage-cost function $g_k(x,u)$, $\forall x \in \mathcal{X}$, $u \in \mathcal{U}$, $k \in \{1, 2, \dots, K\}$.

Specifically, the entry `G{k}(i,j)` of the matrix `G{k}` is the per-stage cost $g_k(i,j)$.

Construct the cost-to-go $V_k(x)$ using the **MATLAB®** code:

```
V{K+1} = zeros(size(G{K},1),1);
for k = K:-1:1
  V{k} = min(G{k}+V{k+1}(F{k}),[],2);
end
```

`[],2` in the `min` function: minimization performed along the second dimension of the matrix (i.e., along the columns).

Solving Finite One-Player Games with MATLAB

Running this code, the following variable is created:

`V` : a cell-array with $K+1$ elements, each equal to an $n_{\mathcal{X}} \times 1$ column vector so that `V{k}` represents the cost-to-go $V_k(x)$, $\forall x \in \mathcal{X}$, $k \in \{1, 2, \ldots, K\}$.

Specifically, the entry `V{k}(i)` of the vector `V{k}` is the cost-to-go $V_k(i)$ from state `i` at stage `k`.

For a given state x at stage k, the optimal action u given by

$$\gamma_k^{\mathrm{FB}}(x_k) := \underset{u_k \in \mathcal{U}_K}{\arg\min} \underbrace{\Big(g_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k)) \Big)}_{\text{computed using the measured state } x_K}, \quad \forall k \in \{1, 2, \ldots, K\}$$

can be obtained using

```
[~,u] = min(G{k}(x,:)+V{k+1}(F{k}(x,:))',[],2);
```

Linear Quadratic Dynamic Games

Linear Quadratic Dynamic Games

Discrete-time **linear quadratic one-player games** are characterized by linear dynamics of the form

$$x_{k+1} = \underbrace{Ax_k + Bu_k}_{f_k(x_k,u_k)}, \quad x \in \mathbb{R}^n, u \in \mathbb{R}^n, k \in \{1, 2, \ldots, K\}$$

and a stage-additive quadratic cost of the form

$$J := \sum_{k=1}^{K} \Big(\underbrace{\|y_k\|^2 + u_k{}' R u_k}_{g_k(x_k,u_k)} \Big) = \sum_{k=1}^{K} \Big(\underbrace{x_k{}' C' C x_k + u_k{}' R u_k}_{g_k(x_k,u_k)} \Big)$$

where

$$y_k = Cx_k, \quad \forall k \in \{1, 2, \ldots, K\}$$

Linear Quadratic Dynamic Games

The cost function

$$J := \sum_{k=1}^{K} \Big(\underbrace{\|y_k\|^2 + {u_k}' R u_k}_{g_k(x_k, u_k)} \Big) = \sum_{k=1}^{K} \Big(\underbrace{{x_k}' C' C x_k + {u_k}' R u_k}_{g_k(x_k, u_k)} \Big)$$

captures scenarios in which the (only) player wants to make the y_k, $k \in \{1, 2, \ldots, K\}$ small without **spending** much effort in their action u_k.

Symmetric positive definite matrix R: a conversion factor that maps units of u_k into units of y_k.

Theorem 15.1 can be used to compute optimal policies for this game and leads to the following result.

Linear Quadratic Dynamic Games

Corollary 15.1. Suppose we define the matrices P_k according to the (backwards) recursion:

$$P_{K+1} = 0$$
$$P_k = C'C + A'P_{k+1}A - A'P_{k+1}B(R + B'P_{k+1}B)^{-1}B'P_{k+1}A$$

$\forall k \in \{1, 2, \ldots, K\}$, and that

$$R + B'P_{k+1}B \geq 0, \quad \forall k \in \{1, 2, \ldots, K\}$$

Then the state-FB policy

$$\gamma_k^{\mathbf{FB}}(x_k) = -(R + B'P_{k+1}B)^{-1}B'P_{k+1}A, \quad \forall k \in \{1, 2, \ldots, K\}$$

is an optimal (time-consistent) state-FB policy for the linear quadratic (LQ) one-player game, leading to an optimal cost equal to $x_1 P_1 x_1$.

Linear Quadratic Dynamic Games

Notation: The equation

$$\begin{aligned} P_{K+1} &= 0 \\ P_k &= C'C + A'P_{k+1}A - A'P_{k+1}B(R + B'P_{k+1}B)^{-1}B'P_{k+1}A \end{aligned}$$

$\forall k \in \{1, 2, \dots, K\}$, is called a **difference Riccati equation**.

Practice Exercise

Practice Exercise

15.1. Prove Corollary 15.1.

Hint: Try to find a solution to

$$V_k(x) = \begin{cases} 0 & k = K+1 \\ \inf_{u_k \in \mathcal{U}_k} \Big(g_k(x, u_k) + V_{k+1}(f_k(x, u_\ell)) \Big) & k \in \{1, 2, \ldots, K\}, \end{cases}$$

$\forall x \in \mathcal{X}$, of the form $V_k(x) = x' P_k x$, $\forall x \in \mathbb{R}^n$, $k \in \{1, 2, \ldots, K+1\}$ for appropriately selected symmetric $n \times n$ matrices P_k.

Practice Exercise

Solution to Exercise 15.1. For this game, $V_k(x)$ is given by

$$V_k(x) = \begin{cases} 0 & k = K+1 \\ \min_{u_k \in \mathbb{R}^{n_u}} \left(x'C'Cx + u_k'Ru_k + V_{k+1}(Ax + Bu_k) \right) & k \in \{1, 2, \ldots, K\} \end{cases}$$

$\forall x \in \mathbb{R}^n$. Inspired by the quadratic form of the per-stage cost, we will try to find a solution to $V_k(x)$ of the form

$$V_k(x) = x'P_k x, \quad \forall x \in \mathbb{R}^n, \quad k \in \{1, 2, \ldots, K+1\}$$

for appropriately selected symmetric $n \times n$ matrices P_k. For $V_k(x)$ to hold, we need to have $P_{K+1} = 0$ and

$$\begin{aligned} x'P_k x &= \min_{u_k \in \mathbb{R}^{n_u}} \left(x'C'Cx + u_k{}'Ru_k + (Ax + Bu_k)'P_{k+1}(Ax + Bu_k) \right) \\ &= \min_{u_k \in \mathbb{R}^{n_u}} \left(x'(C'C + A'P_{k+1}A)x + u_k{}'(R + B'P_{k+1}B)u_k + 2x'A'P_{k+1}Bu_k \right) \end{aligned}$$

$\forall x \in \mathbb{R}^n, k \in \{1, 2, \ldots, K\}$.

Practice Exercise

Since the function to optimize is quadratic, to compute the minimum in $x'P_kx$ we simply need to make the appropriate gradient equal to zero:

$$\frac{\partial}{\partial u_k}\Big(x'(C'C + A'P_{k+1}A)x + u_k{}'(R + B'P_{k+1}B)u_k + 2x'A'P_{k+1}Bu_k\Big) = 0$$
$$\Leftrightarrow 2u_k{}'(R + B'P_{k+1}B) + 2x'A'P_{k+1}Bu_k = 0$$
$$\Leftrightarrow u_k = -(R + B'P_{k+1}B)^{-1}B'P_{k+1}Ax$$

Therefore

$$\min_{u_k \in \mathbb{R}^{n_u}} \Big(\underbrace{x'(C'C + A'P_{k+1}A)x + u_k{}'(R + B'P_{k+1}B)u_k + 2x'A'P_{k+1}Bu_k}_{u_k = -(R+B'P_{k+1}B)^{-1}B'P_{k+1}Ax}\Big)$$
$$= x'(C'C + A'P_{k+1}A - A'P_{k+1}B(R + B'P_{k+1}B)^{-1}B'P_{k+1}A)x$$

Practice Exercise

This means that $x'P_kx$ is of the form

$$x'P_kx = x'(C'C + A'P_{k+1}A - A'P_{k+1}B(R + B'P_{k+1}B)^{-1}B'P_{k+1}A)x$$

which holds in view of the **difference Riccati equation**.

Corollary 15.1 then follows directly from **Theorem 15.1**, since we have found a sequence of functions $V_1(x), V_2(x), \ldots, V_{K+1}(x)$ that satisfies $V_k(x)$ for which the infimum is always achieved at the point u_k given by making the appropriate gradient equal to zero.

Note. The value for the minimum will provide the optimal policy.

End of Lecture

15 - One-Player Dynamic Games

Questions?