# نظریه بازیها
# Game Theory

ارائه کننده: امیرحسین نیکوفرد
مهندسی برق و کامپیوتر دانشگاه خواجه نصیر

دانشگاه صنعتی خواجه نصیرالدین طوسی

# InfiniteDynamic Games

Material

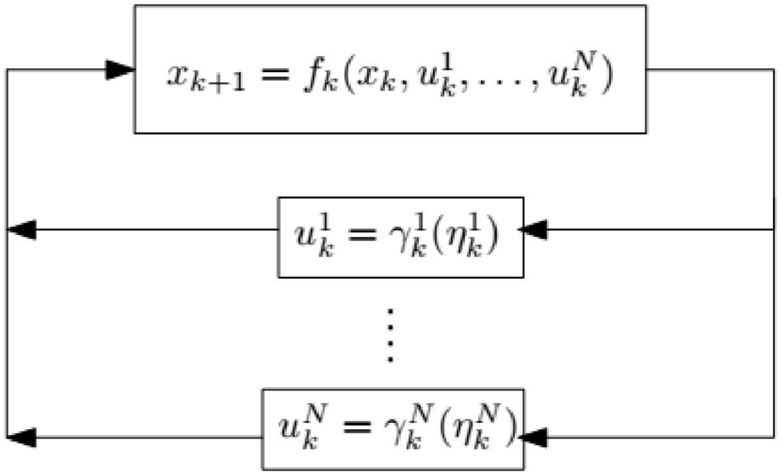
- Dynamic Non-cooperative Game Theory: Second Edition
  - Chapter5: Sections 5:1–5:3.

## InfiniteDynamic Games

- ❑ Zero sum games
- ❑ Non-zero sum games
- ❑ Infinite Games
- ❑ **Infinite Dynamic Games**
  - ❑ Dynamic games in discrete time
  - ❑ Information structures
  - ❑ **Continuous-time differential games**
  - ❑ **Discrete-time dynamic programming**

## Loop model (recap)

- Basic elements of the loop model:
  - Set of Players: $P_i, i \in [1,\ldots,N]$
  - Stages: $k \in K, K := [1,\ldots,K]$
  - State Space: $x_k \in X$
  - Action Space: $u_k^i \in U_k^i, u_k^{-i} \in U_k^{-i}$


$x_{k+1} = f_k(x_k, u_k^1, \ldots, u_k^N)$
$u_k^1 = \gamma_k^1(\eta_k^1)$
$u_k^N = \gamma_k^N(\eta_k^N)$

- Functionals in the loop model:
  - State Equation: Describes the state evolution

$$x_{k+1} = f_k(x_k, u_k^N, \ldots, u_k^N), \quad k \in K, x_k \in X$$

  - Strategies: $P_i$ uses strategy $\gamma^i = \{\gamma_1^i, \ldots, \gamma_K^i\}$, *for* $\gamma^i \in \Gamma^i$

$$u_k^i = \gamma_k^i(\eta_k^i); \quad \gamma_k^i \in \Gamma_k^i$$

  - Cost Functional: The real-valued cost for $P_i$ is given by

$$L^i(x_1, u_1^1, \ldots, u_1^N; \ldots; x_k, u_K^1, \ldots, u_K^N)$$

## Examples of Information Structures

A few typical information structures (IS):

- Open Loop IS: $\eta_k^i = x_1 , k \in K$
- Closed-Loop Perfect State IS: $\eta_k^i = \{x_1,\ldots,x_k\} , k \in K$
- Closed-Loop Imperfect State IS: $\eta_k^i = \{y_1^i,\ldots,y_k^i\} , k \in K$
- Memoryless IS: $\eta_k^i = \{x_1, x_k\} , OR \quad \eta_k^i = \{y_1^i, y_k^i\} , k \in K$
- Feedback IS: $\eta_k^i = x_k , OR \quad \eta_k^i = y_k^i , k \in K$
- One step delayed IS:

$$\eta_k^i = \{x_1,\ldots,x_{k-1}\} , OR \quad \eta_k^i = \{y_1^i,\ldots,y_{k-1}^i\} , k \in K$$

## Continuous-time differential games

Dynamic games are often formulated also in continuous time, which means that

- the state *x(t)* varies continuously with time on a given interval $t \in [0, \mathrm{T}]$, and
- the players continuously select actions $u_i(t)$ on $[0, \mathrm{T}]$, which determine the state's evolution.

When the state x(t) is an n-vector of real numbers whose evolution is determined by a differential equation, the game is called a differential game. We consider here differential games with dynamics of the form

$$\underbrace{\dot{x}}_{\substack{\textit{state}\\\textit{derivative}}} = \underbrace{f}_{\substack{\textit{game}\\\textit{dynamics}}} \big( \underbrace{t}_{\textit{time}}, \underbrace{x(t)}_{\substack{\textit{current}\\\textit{state}}}, \underbrace{u_i(t)}_{\substack{P_i\textit{'s action}\\\textit{at time } t}} \big), \qquad \forall \mathrm{t} \in [0, \mathrm{T}] \qquad (1)$$

## Continuous-time differential games

for which each player $P_i$, $i \in [1,2,\dots,N]$ wants to minimize a cost of the form

$$J_i = \underbrace{\int_0^T g_i(t, x(t), u_i(t))dt}_{\text{cost } along\ trajctory} + \underbrace{q_i(x(T))}_{final \text{ cost}} \qquad (2)$$

For such games we shall also consider **open-loop** policies of the form

$$u_i(t) = \gamma^{OL}(t, x(0)) \qquad \forall \mathrm{t} \in [0, \mathrm{T}]$$

and **(perfect) state-feedback** policies of the form

$$u_i(t) = \gamma^{FB}(t, x(t)) \qquad \forall \mathrm{t} \in [0, \mathrm{T}]$$

## Example (zebra in the lake)

**Example:** Consider the zebra in the lake game where

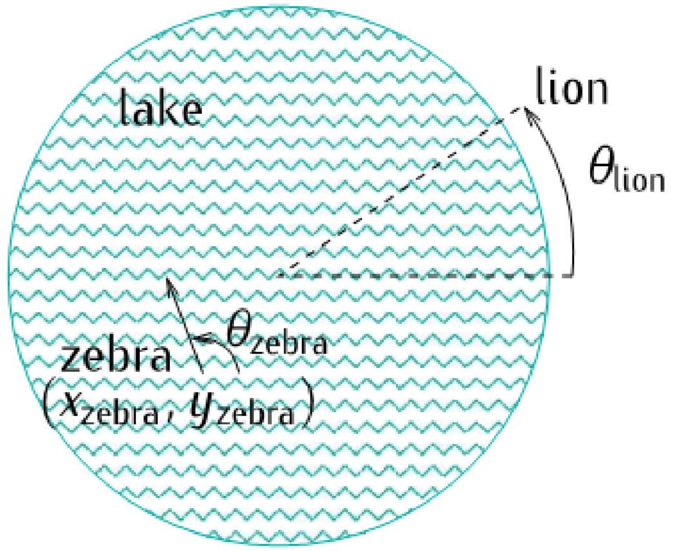
- the player $P_1$ is a *zebra* that swims with a speed of $V_{zebra}$ in a circular lake with radius *R*, and
- the player $P_2$ is a *lion* that runs along the perimeter of the lake with maximum speed of $V_{lion} > V_{zebra}$ .

**Objective:** The two players have opposite objectives:

1. The zebra wants to get to the shore of the lake without being caught coming out of the water.
2. The lion wants to be at the precise position where the zebra leaves the lake.

## Example (zebra in the lake)

**Policies:** In this game it is assumed that each player constantly sees the other and can react instantaneously to the current position/velocity of the other player. This game only makes sense in closed loop if the lion can see the zebra and uses this to decide where to run, because otherwise the lion has no chance of ever catching the zebra.



## Example (zebra in the lake)

- Denoting by $(X_{zebra}, Y_{zebra})$ the position of the zebra and by $\theta_{zebra}$ zebra the orientation, we have that

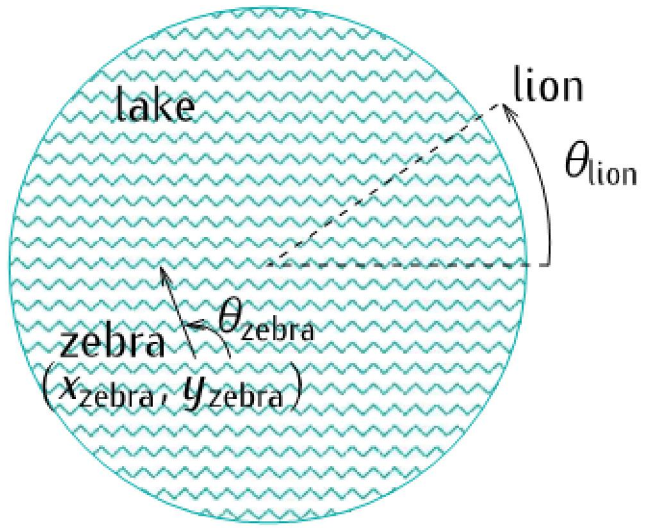
$$\dot{x}_{zebra} = v_{zebra}\cos\theta_{zebra}, \qquad \dot{y}_{zebra} = v_{zebra}\sin\theta_{zebra}, \qquad \theta_{zebra} \in [0, 2\pi)$$

- and denoting by $\theta_{lion}$ and $\omega_{lion}$ the (angular) position and velocity of the lion, respectively, we have that

$$\dot{\theta}_{lion} = \omega_{lion} \qquad \omega_{lion} \in [-\frac{v_{lion}}{R}, \frac{v_{lion}}{R})$$

- Defining a state vector

$$x(t) := [x_{zebra}, y_{zebra}, \theta_{lion}]^T$$



## Example (zebra in the lake)

- the equations of position the zebra and lion can be written as in (1), where the actions of the players are:

$$u(t) = \theta_{zebra} \in [0, 2\pi) \qquad d(t) = \omega_{lion} \in [-\frac{v_{lion}}{R}, \frac{v_{lion}}{R})$$

- If we assume that the zebra wants to get out of the lake as soon as possible without being captured, the zebra's cost is of the form

$$J_1 = \begin{cases} T_{exit} & \textit{zebra exists the lake safely at time } T_{exit} \\ \infty & \textit{zebra gets caught when she exists.} \end{cases}$$

- This is a zero-sum game and therefore the lion wants to maximize $J_1$, or equivalently minimize $J_2 := -J_1$.

## Example (zebra in the lake)

- A common trick that is used to write such a cost as in an integral form such as (2) is to freeze the state when the zebra reaches the shore, which amounts to replacing state space
- This is a zero-sum game and therefore the lion wants to maximize $J_1$, or equivalently minimize $J_2 := -J_1$.

$$\begin{bmatrix} \dot{x}_{zebra} \\ \dot{y}_{zebra} \\ \dot{\theta}_{lion} \end{bmatrix} = \begin{cases} \begin{bmatrix} v_{zebra}\cos\theta_{zebra} \\ v_{zebra}\sin\theta_{zebra} \\ \omega_{lion} \end{bmatrix} & x^2_{zebra} + y^2_{zebra} < R^2 \\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} & x^2_{zebra} + y^2_{zebra} = R^2 \end{cases}$$

## Example (zebra in the lake)

- and then defining

$$J_1 = \int_0^\infty g\,(x_{zebra}, y_{zebra}, \theta_{lion})dt$$

Where

$$g\,(x_{zebra}, y_{zebra}, \theta_{lion}) = \begin{cases} 1 & x^2_{zebra} + y^2_{zebra} < R^2 \\ 1 & x_{zebra} = R\cos\theta_{lion}, \quad y_{zebra} = R\sin\theta_{lion} \\ 0 & otherwise \begin{pmatrix} zebra\ reaches\ shore\ away \\ from\ lion \end{pmatrix} \end{cases}.$$

- This game is only meaningful in the context of state-feedback policies, because the lion basically has no chance of capturing the zebra unless the lion can see the zebra.

## InfiniteDynamic Games

- ❑ Zero sum games
- ❑ Non-zero sum games
- ❑ Infinite Games
- ❑ **Infinite Dynamic Games**
    - ❑ Dynamic games in discrete time
    - ❑ Information structures
    - ❑ Continuous-time differential games
    - ❑ **Discrete-time dynamic programming**

## Dynamic games in discrete time

**One-player discrete-time games:**

We start by discussing solution methods for one-player dynamic games, which are simple optimizations. In the context of discrete-time dynamic games, this corresponds to dynamics of the form

$$\underbrace{x_{k+1}}_{\substack{state\\ at\ stage\ k+1}} = \underbrace{f_k}_{\substack{\text{"}dynamics\text{"}\ at\\ stage\ k}} ( \underbrace{x_k}_{\substack{state\\ at\ stage\ k}} , \underbrace{u_k}_{\substack{P_1\text{'}s\ action\\ at\ stage\ k}} ) \quad ,\forall \mathrm{k} \in \{1,2,\ldots,\mathrm{K}\}, \qquad (3)$$

We assume a finite horizon $(K < \infty)$ stage additive costs of the form

$$J = \sum_{k=1}^{K} \mathrm{g}(\mathrm{x}_k , \mathrm{u}_k) \qquad (4)$$

## Dynamic games in discrete time

that the (only) player wants to minimize using either a open-loop policy

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

or a state-feedback policy

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

### Discrete-time cost-to-go

Suppose that the player finds herself at some state *x* at stage *l*. This state would perhaps not be the optimal place to be at this stage, but nevertheless she would like to estimate the minimum cost that she should expect, if she were to play optimally from this point on so as to minimize the costs incurred in the remaining stages.

## Discrete-time cost-to-go

This scenario motivates defining the cost-to-go from state $x \in X$ at time $l \in \{1, 2, \ldots, \mathrm{K}\}$ by

$$V_l(x) := \inf \sum_{k=l}^{K} g(\mathrm{x}_k, \mathrm{u}_k), \qquad \forall x \in X \qquad (5)$$

with the sequence $\{x_k \in X : \mathrm{k} = l, l+1, \ldots, \mathrm{K}\}$ starting at

$$x_l = x$$

and satisfying the dynamics

$$x_{k+1} = f(x_k, u_k), \qquad \forall \mathrm{k} \in \{l, l+1, \ldots, \mathrm{K}-1\},$$

## Discrete-time cost-to-go

- Computing the cost-to-go $V_1(x_1)$ from the initial state $x_1$ at the first stage $l = 1$ essentially amounts to minimizing the cost (4) for the dynamics (3). This observation leads to two important conclusions:

1) Regardless of the information structure considered (open loop, state feedback, or other), it is not possible to obtain a cost (4) lower than $V_1(x_1)$ .This is because in the minimization in (5) we place no constraints on what information may or may not be available to compute the optimal $u_k$

2) If the infimum in (5) is achieved for some specific sequence

$$u_1^* \in U_1, u_2^* \in U_2, \ldots, u_K^* \in U_K,$$

## Discrete-time cost-to-go

that can be computed before the game starts just with knowledge of $x_1$, then this sequence of actions provides an optimal ***open-loop*** **policy** $\gamma_k^{(OL)}$

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

In this case, $V_1\ (x_1)$ is precisely equal to the smallest value that can be achieved for (4).

## Discrete-time dynamic programming

### Discrete-time dynamic programming

- Dynamic programming is a computationally efficient recursive technique that can be used to compute the cost-to-go. For the last stage *K*, the cost-to-go $V_K(x)$ is simply the minimum of

$$g_K(\mathrm{x}_K, \mathrm{u}_K)$$

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

$$V_K(x) = \inf_{u_K \in U_K} \quad g_K(\mathrm{x}, \mathrm{u}_K), \qquad \forall \mathrm{x} \in \mathrm{X}$$

## Discrete-time dynamic programming

- Therefore, for each state *x*, we can compute $V_K(x)$ by solving a single-parameter optimization over the set $U_K$. For the previous stages $l < K$, we have that

$$V_l(x) = \inf_{u_l \in U_l, \ldots u_K \in U_K} \sum_{k=l}^{K} g_k(\mathrm{x}_k, \mathrm{u}_K)$$

$$= \inf_{u_l \in U_l, \ldots u_K \in U_K} (g_l(\mathrm{x} \;, \mathrm{u}_l) \quad + \sum_{k=l+1}^{K} g_k(\mathrm{x}_k, \mathrm{u}_K))$$

$$= \inf_{u_l \in U_l} (g_l(\mathrm{x} \;, \mathrm{u}_l) \quad + \inf_{u_l+1 \in U_l+1, \ldots u_K \in U_K} \sum_{k=l+1}^{K} g_k(\mathrm{x}_k, \mathrm{u}_K))$$

## Discrete-time dynamic programming

where in the first equality we used the fact that we must set $x_l = x$ to compute $V_l(x)$ and in the second we used the fact that $g_l(x, u_l)$ does not depend on $u_{l+1}, \ldots, u_K$. However

$$\inf_{u_l \in U_l + 1, \ldots u_K \in U_K} \sum_{k=l+1}^{K} g_k(x_k, u_K)$$

is precisely the minimum cost for a game starting at stage $l + 1$ with the state $x_{l+1} = f_l(x, u_l)$

which is precisely the cost-to-go $V_{l+1}(f_l(x,u_l))$. We therefore conclude that

$$V_l(x) = \inf_{u_l \in U_l} (g_l(x\ ,u_l) + V_{l+1}(f_l(x,u_l)), \quad \forall x \in X, \quad l \in \{1,2,\ldots,K-1\} \quad (6)$$

## Discrete-time dynamic programming

- This shows that if we know the function $V_{l+1}$, then we can compute each $V_l(x)$ by solving a single-parameter optimization over the set $U_l$. Moreover, this optimization, produces the optimal action $u_l^*$ to be used when the state is at $x_l$.
- It is convenient to define

$$V_{K+1}(x) = 0, \quad \forall \mathrm{x} \in \mathrm{X}$$

which allow us to re-write both (5) and (6) using the formula:

$$V_l(x) = \inf_{u_l \in U_l} (g_l(\mathrm{x}\ ,\mathrm{u}_l) + V_{l+1}(f_l(x,u_l)), \quad \forall \mathrm{x} \in \mathrm{X}, \quad l \in \{1,2,\ldots,K\}$$