Robotic Systems II

Lecture 4: Deterministic Optimal Control & LQR

Konstantinos Chatzilygeroudis - costashatz@upatras.gr

Department of Electrical and Computer Engineering
University of Patras

Template made by Panagiotis Papagiannopoulos

Deterministic Optimal Control

Continuous Time:

$$\underset{\boldsymbol{x}(t),\boldsymbol{u}(t)}{\operatorname{argmin}} \mathcal{J}(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \int_{t_0}^{t_f} \ell(\boldsymbol{x}(t), \boldsymbol{u}(t))dt + \ell_F(\boldsymbol{x}(t_f))$$

$$\text{s.t.} \qquad \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t))$$

where

- $\boldsymbol{x}(t) \in \mathbb{R}^N, \boldsymbol{u}(t) \in \mathbb{R}^M$ are the state and control trajectories
- $\mathcal{J}(\boldsymbol{x}(t), \boldsymbol{u}(t))$ is the "cost function"
- $\ell(\boldsymbol{x}(t), \boldsymbol{u}(t))$ is the "stage cost"
- $\ell_F(\boldsymbol{x}(t_f))$ is the "terminal cost"
- $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t))$ are the "dynamics constraints"
- We can potentially add more constraints (e.g. torque limits)

Deterministic Optimal Control (2)

- The continuous version is **“infinite dimensional”**

Deterministic Optimal Control (2)

- The continuous version is **“infinite dimensional”**
- The solution is open loop control trajectories; we know everything!
- Only very few problems can be solved analytically

Deterministic Optimal Control (2)

- The continuous version is **“infinite dimensional”**
- The solution is open loop control trajectories; we know everything!
- Only very few problems can be solved analytically
- Let’s discretize!

Discrete Time:

$$\underset{\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}}{\text{argmin}} \mathcal{J}(\boldsymbol{x}_{1:K}, \boldsymbol{u}_{1:K-1}) = \sum_{k=1}^{K-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \ell_F(\boldsymbol{x}_K)$$

$$\text{s.t.} \qquad \boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

- $\boldsymbol{x}_k \in \mathbb{R}^N$ and $\boldsymbol{u}_k \in \mathbb{R}^M$ are vectors
- The is now **“finite dimensional”**
- The solution is still open loop control trajectories; we know everything!
- We usually call $\boldsymbol{x}_k, \boldsymbol{u}_k$ **“knot points”**

Pontryagin's Minimum Principle basically refers to the **KKT conditions** for the optimal control problem. We will skip the derivation here and focus on the result:

$$
\begin{aligned}
\boldsymbol{x}_{k+1} &= \nabla_{\boldsymbol{\lambda}} H(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\lambda}_{k+1}) \\
\boldsymbol{\lambda}_k &= \nabla_{\boldsymbol{x}} H(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\lambda}_{k+1}) \\
\boldsymbol{u}_{k+1} &= \underset{\boldsymbol{u}}{\operatorname{argmin}}\, H(\boldsymbol{x}_k, \boldsymbol{u}, \boldsymbol{\lambda}_{k+1}), \text{s.t. } \boldsymbol{u} \in \mathcal{U} \\
\boldsymbol{\lambda}_K &= \frac{\partial \ell_F}{\partial \boldsymbol{x}_K}
\end{aligned}
$$

where $H(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\lambda}_{k+1}) = \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{\lambda}_{k+1}^T f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k)$.

What is the Linear Quadratic Regulator (LQR) problem?

What is the Linear Quadratic Regulator (LQR) problem?

$$\underset{\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}}{\operatorname{argmin}} \mathcal{J}(\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}) = \sum_{k=1}^{K-1}\left(\frac{1}{2}\boldsymbol{x}_k^T\boldsymbol{Q}_k\boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T\boldsymbol{R}_k\boldsymbol{u}_k\right) + \frac{1}{2}\boldsymbol{x}_K^T\boldsymbol{Q}_K\boldsymbol{x}_K$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k$$

$$\boldsymbol{Q}_k \succeq 0$$

$$\boldsymbol{R}_k \succ 0$$

- Widely used in many real applications
- The “workhorse” of optimal control
- We know everything about it!
- Infinite variations and extensions!
- **Time Invariant** if: $\boldsymbol{A}_k = \boldsymbol{A}, \boldsymbol{B}_k = \boldsymbol{B}, \boldsymbol{Q}_k = \boldsymbol{Q}, \boldsymbol{R}_k = \boldsymbol{R}, \forall k$

LQR via Shooting

We apply Pontryagin's Minimum Principle to the specific LQR problem (time invariant case for simplicity):

LQR via Shooting

We apply Pontryagin's Minimum Principle to the specific LQR problem (time invariant case for simplicity):

$$
\begin{aligned}
\boldsymbol{x}_{k+1} &= \nabla_{\boldsymbol{\lambda}} H(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\lambda}_{k+1}) = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \\
\boldsymbol{\lambda}_k &= \nabla_{\boldsymbol{x}} H(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\lambda}_{k+1}) = \boldsymbol{Q}\boldsymbol{x}_k + \boldsymbol{A}^T\boldsymbol{\lambda}_{k+1} \\
\boldsymbol{u}_{k+1} &= \operatorname*{argmin}_{\boldsymbol{u}} H(\boldsymbol{x}_k, \boldsymbol{u}, \boldsymbol{\lambda}_{k+1}) = -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{\lambda}_{k+1} \\
\boldsymbol{\lambda}_K &= \frac{\partial \ell_F}{\partial \boldsymbol{x}_K} = \boldsymbol{Q}_K\boldsymbol{x}_K
\end{aligned}
$$

LQR via Shooting (2)

1. We start with a guess of the trajectory of $\boldsymbol{u}_{1:K-1}$
2. Forward pass (rollout) given $\boldsymbol{u}_{1:K-1}$ and $\boldsymbol{x}_1$ to get $\boldsymbol{x}_{1:K}$
3. Backward pass to compute $\boldsymbol{\lambda}_{1:K}$ and $\boldsymbol{u}_{1:K-1}$ (or better $\Delta\boldsymbol{u}_{1:K-1}$)
4. Forward pass (rollout) with line search on $\Delta\boldsymbol{u}_{1:K-1}$ to get new $\boldsymbol{x}_{1:K}$
5. Go back to 3 until convergence

Double Integrator:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}$$

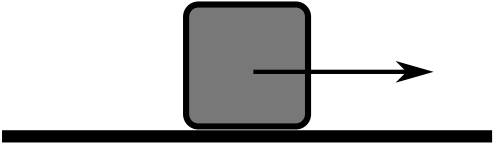
$$\boldsymbol{u} = K\ddot{\boldsymbol{q}}$$

where

$$\boldsymbol{q} = \begin{bmatrix} x \end{bmatrix} \in \mathbb{R}$$

$$\dot{\boldsymbol{q}} = \begin{bmatrix} v \end{bmatrix} = \begin{bmatrix} \dot{x} \end{bmatrix} \in \mathbb{R}$$

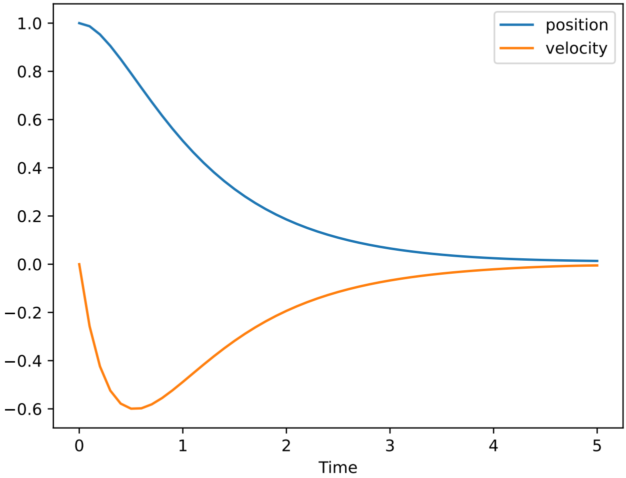
$$\boldsymbol{u} = \begin{bmatrix} K\ddot{x} \end{bmatrix} \in \mathbb{R}$$



Discrete Dynamics:

$$\boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k) = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} \frac{1}{2K} dt^2 \\ \frac{dt}{K} \end{bmatrix} \boldsymbol{u}_k$$

LQR via Shooting - Code Example



LQR as a Quadratic Programming Problem

Looking at the LQR problem, it really looks like a QP. Can we write it like one?

Looking at the LQR problem, it really looks like a QP. Can we write it like one? Yes we can! We assume that $\boldsymbol{x}_1$ (initial conditions) is given, and define:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{x}_K \end{bmatrix}, \boldsymbol{H} = \begin{bmatrix} \boldsymbol{R}_1 & & & & \\ & \boldsymbol{Q}_2 & & & \\ & & \boldsymbol{R}_2 & & \\ & & & \ddots & \\ & & & & \boldsymbol{Q}_K \end{bmatrix}$$

Now we can define:

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z}$$
$$\text{s.t.} \qquad \text{"dynamics constraints"}$$

For the dynamics constraints we have:

$$\underbrace{\begin{bmatrix} \boldsymbol{B}_1 & -\boldsymbol{I} & \boldsymbol{0} & \dots & & & & \\ \boldsymbol{0} & \boldsymbol{A}_2 & \boldsymbol{B}_2 & -\boldsymbol{I} & \boldsymbol{0} & \dots & & \\ & & & \ddots & & & & \\ & & & & & \boldsymbol{A}_{K-1} & \boldsymbol{B}_{K-1} & -\boldsymbol{I} \end{bmatrix}}_{\boldsymbol{G}} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{x}_K \end{bmatrix} = \underbrace{\begin{bmatrix} -\boldsymbol{A}_1\boldsymbol{x}_1 \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}}_{\boldsymbol{d}}$$

Now we have a full QP:

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z}$$

$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\operatorname{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z}$$
$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this?

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z}$$
$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this? Of course we can! The Lagrangian is:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} + \boldsymbol{\lambda}^T(\boldsymbol{G}\boldsymbol{z} - \boldsymbol{d})$$

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H}\boldsymbol{z}$$

$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this? Of course we can! The Lagrangian is:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H}\boldsymbol{z} + \boldsymbol{\lambda}^T(\boldsymbol{G}\boldsymbol{z} - \boldsymbol{d})$$

KKT Conditions:

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \boldsymbol{H}\boldsymbol{z} + \boldsymbol{G}^T\boldsymbol{\lambda} = \boldsymbol{0}$$
$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H} \boldsymbol{z}$$
$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this? Of course we can! The Lagrangian is:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H}\boldsymbol{z} + \boldsymbol{\lambda}^T(\boldsymbol{G}\boldsymbol{z} - \boldsymbol{d})$$

KKT Conditions:

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \boldsymbol{H}\boldsymbol{z} + \boldsymbol{G}^T\boldsymbol{\lambda} = \boldsymbol{0}$$
$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

KKT System:

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{G}^T \\ \boldsymbol{G} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{d} \end{bmatrix}$$

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\text{argmin}}\,\mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z}$$
$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this? Of course we can! The Lagrangian is:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} + \boldsymbol{\lambda}^T(\boldsymbol{G}\boldsymbol{z} - \boldsymbol{d})$$

KKT Conditions:

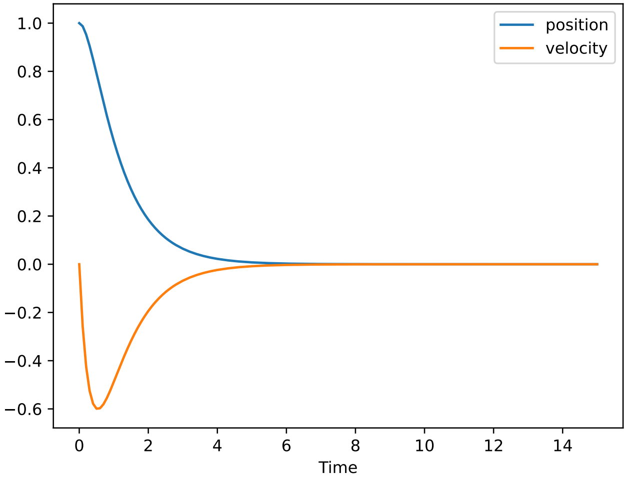
$$\nabla_{\boldsymbol{z}}\mathcal{L} = \boldsymbol{H}\boldsymbol{z} + \boldsymbol{G}^T\boldsymbol{\lambda} = \boldsymbol{0}$$
$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

KKT System:

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{G}^T \\ \boldsymbol{G} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{d} \end{bmatrix}$$

We have the exact solution by solving ONE linear system!

LQR as a QP - Code Example



Let's write the KKT System in "detail" (for $K = 4$):

$$
\left[\begin{array}{cccccccc}
R_1 & 0 & \cdots & & 0 & B_1^T & \cdots & 0 \\
0 & Q_2 & 0 & \cdots & 0 & -I & A_2^T & \vdots \\
\vdots & 0 & R_2 & 0 & \cdots & 0 & B_2^T & \\
& \vdots & 0 & Q_3 & 0 & \cdots & -I & A_3^T \\
& & \vdots & 0 & R_3 & 0 & \cdots & B_3^T \\
0 & 0 & & \vdots & 0 & Q_4 & & -I \\
\hdashline
B_1 & -I & 0 & 0 & \vdots & 0 & \ddots & \vdots \\
\vdots & A_2 & B_2 & -I & 0 & \vdots & & \\
0 & \cdots & & A_3 & B_3 & -I & \cdots & 0
\end{array}\right]
\left[\begin{array}{c}
u_1 \\ x_2 \\ u_2 \\ x_3 \\ u_3 \\ x_4 \\ \hdashline \lambda_2 \\ \lambda_3 \\ \lambda_4
\end{array}\right]
=
\left[\begin{array}{c}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \hdashline -A_1 x_1 \\ 0 \\ 0
\end{array}\right]
$$

LQR via Riccati Recursion (2)

We first solve the last line of the upper block:

$$\boldsymbol{Q}_4\boldsymbol{x}_4 - \boldsymbol{\lambda}_4 = \boldsymbol{0} \Rightarrow \boldsymbol{\lambda}_4 = \boldsymbol{Q}_K\boldsymbol{x}_4$$

Then we move one line up:

$$\begin{aligned}&\boldsymbol{R}_3\boldsymbol{u}_3 + \boldsymbol{B}_3^T\boldsymbol{\lambda}_4 = \boldsymbol{R}_3\boldsymbol{u}_3 + \boldsymbol{B}_3^T\boldsymbol{Q}_K\boldsymbol{x}_4 = \boldsymbol{0}\\ &\Rightarrow \boldsymbol{R}_3\boldsymbol{u}_3 + \boldsymbol{B}_3^T\boldsymbol{Q}_K(\boldsymbol{A}_3\boldsymbol{x}_3 + \boldsymbol{B}_3\boldsymbol{u}_3) = \boldsymbol{0}\\ &\Rightarrow \boldsymbol{u}_3 = -\underbrace{(\boldsymbol{R}_3 + \boldsymbol{B}_3^T\boldsymbol{Q}_K\boldsymbol{B}_3)^{-1}\boldsymbol{B}_3^T\boldsymbol{Q}_K\boldsymbol{A}_3}_{\boldsymbol{K}_3}\boldsymbol{x}_3\end{aligned}$$

One more line:

$$\begin{aligned}&\boldsymbol{Q}_3\boldsymbol{x}_3 - \boldsymbol{\lambda}_3 + \boldsymbol{A}_3^T\boldsymbol{\lambda}_4 = \boldsymbol{0} \Rightarrow \boldsymbol{Q}_3\boldsymbol{x}_3 - \boldsymbol{\lambda}_3 + \boldsymbol{A}_3^T\boldsymbol{Q}_K\boldsymbol{x}_4 = \boldsymbol{0}\\ &\Rightarrow \boldsymbol{Q}_3\boldsymbol{x}_3 - \boldsymbol{\lambda}_3 + \boldsymbol{A}_3^T\boldsymbol{Q}_K(\boldsymbol{A}_3\boldsymbol{x}_3 + \boldsymbol{B}_3\boldsymbol{u}_3) = \boldsymbol{0}\\ &\Rightarrow \boldsymbol{\lambda}_3 = \underbrace{\Big(\boldsymbol{Q}_3 + \boldsymbol{A}_3^T\boldsymbol{Q}_K(\boldsymbol{A}_3 - \boldsymbol{B}_3\boldsymbol{K}_3)\Big)}_{\boldsymbol{P}_3}\boldsymbol{x}_3\end{aligned}$$

LQR via Riccati Recursion (3)

The recursion is now revealed:

$$
\begin{aligned}
\boldsymbol{P}_K &= \boldsymbol{Q}_K \\
\boldsymbol{K}_k &= (\boldsymbol{R}_k + \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{B}_k)^{-1} \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{A}_k \\
\boldsymbol{P}_k &= \boldsymbol{Q}_k + \boldsymbol{A}_k^T \boldsymbol{P}_{k+1} (\boldsymbol{A}_k - \boldsymbol{B}_k \boldsymbol{K}_k)
\end{aligned}
$$

- QP has complexity $O(K^3(N+M)^3)$
- Riccati recursion has complexity $O(K(N+M)^3)$
- The Riccati recursion is the optimal exploitation of the sparsity of the KKT system!

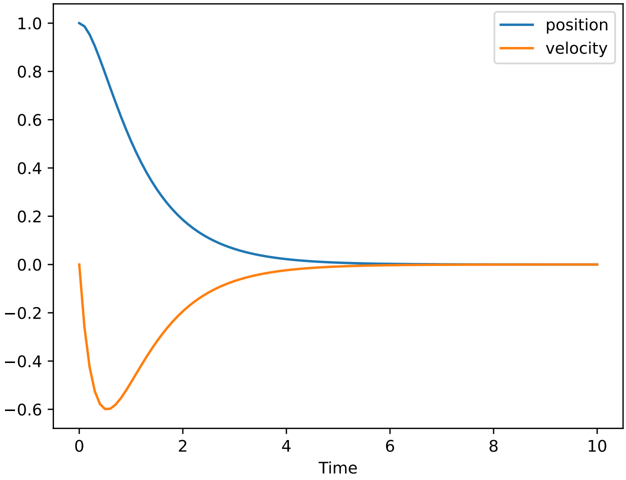
LQR via Riccati Recursion (3)

The recursion is now revealed:

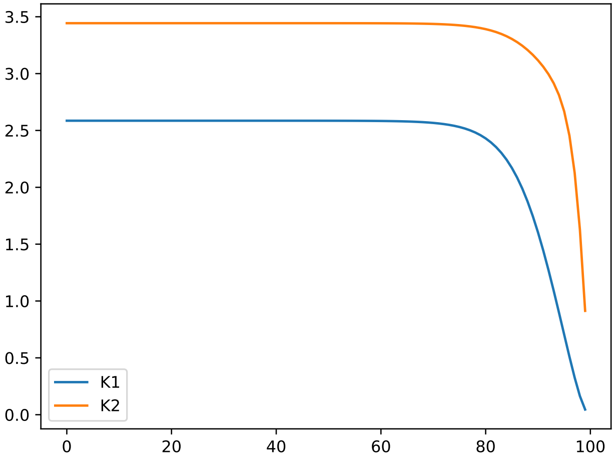
$$\boldsymbol{P}_K = \boldsymbol{Q}_K$$
$$\boldsymbol{K}_k = (\boldsymbol{R}_k + \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{B}_k)^{-1} \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{A}_k$$
$$\boldsymbol{P}_k = \boldsymbol{Q}_k + \boldsymbol{A}_k^T \boldsymbol{P}_{k+1} (\boldsymbol{A}_k - \boldsymbol{B}_k \boldsymbol{K}_k)$$

- QP has complexity $O(K^3(N+M)^3)$
- Riccati recursion has complexity $O(K(N+M)^3)$
- The Riccati recursion is the optimal exploitation of the sparsity of the KKT system!
- **We also get for free a feedback policy: $\boldsymbol{u} = -\boldsymbol{K}\boldsymbol{x}$!**

LQR via Riccati Recursion - Code Example



LQR via Riccati Recursion - Code Example (2)



Infinite Horizon LQR

- Only possible for the time invariant linear system!
- $\boldsymbol{K}_k$ and $\boldsymbol{P}_k$ matrices converge to constant values!
- How can we find those values?
  1. We do the Riccati recursion long enough until they converge!

Infinite Horizon LQR

- Only possible for the time invariant linear system!
- $\boldsymbol{K}_k$ and $\boldsymbol{P}_k$ matrices converge to constant values!
- How can we find those values?
  1. We do the Riccati recursion long enough until they converge!
  2. Let's have a look at the recursion equations again:

$$\boldsymbol{K}_k = (\boldsymbol{R}_k + \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{B}_k)^{-1} \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{A}_k$$

$$\boldsymbol{P}_k = \boldsymbol{Q}_k + \boldsymbol{A}_k^T \boldsymbol{P}_{k+1} (\boldsymbol{A}_k - \boldsymbol{B}_k \boldsymbol{K}_k)$$

Infinite Horizon LQR

- Only possible for the time invariant linear system!
- $\boldsymbol{K}_k$ and $\boldsymbol{P}_k$ matrices converge to constant values!
- How can we find those values?
  1. We do the Riccati recursion long enough until they converge!
  2. Let's have a look at the recursion equations again:

$$\boldsymbol{K}_k = (\boldsymbol{R}_k + \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{B}_k)^{-1} \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{A}_k$$
$$\boldsymbol{P}_k = \boldsymbol{Q}_k + \boldsymbol{A}_k^T \boldsymbol{P}_{k+1} (\boldsymbol{A}_k - \boldsymbol{B}_k \boldsymbol{K}_k)$$

  3. At the limit, we need $\boldsymbol{P}_k = \boldsymbol{P}_{k+1}$ and $\boldsymbol{K}_k = \boldsymbol{K}_{k+1}$
  4. We solve a root finding problem (e.g. with Newton's method)!
- The recursion version is basically the Fixed Point method!

Controllability

How can we know LQR will always find a solution?

- We get to choose $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$
- We cannot choose $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$
- What can we do?

How can we know LQR will always find a solution?

- We get to choose $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$
- We cannot choose $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$
- What can we do? **Let's start with the time invariant case:**

$$
\begin{aligned}
\boldsymbol{x}_K &= \boldsymbol{A}\boldsymbol{x}_{K-1} + \boldsymbol{B}\boldsymbol{u}_{K-1} \\
&= \boldsymbol{A}(\boldsymbol{A}\boldsymbol{x}_{K-2} + \boldsymbol{B}\boldsymbol{u}_{K-2}) + \boldsymbol{B}\boldsymbol{u}_{K-1} \\
&\quad\vdots \\
&= \boldsymbol{A}^K\boldsymbol{x}_1 + \boldsymbol{A}^{K-1}\boldsymbol{B}\boldsymbol{u}_1 + \boldsymbol{A}^{K-2}\boldsymbol{B}\boldsymbol{u}_2 + \cdots + \boldsymbol{B}\boldsymbol{u}_{K-1} \\
&= \underbrace{\begin{bmatrix}\boldsymbol{B} & \boldsymbol{A}^2\boldsymbol{B} & \dots & \boldsymbol{A}^{K-1}\boldsymbol{B}\end{bmatrix}}_{\boldsymbol{C}} \begin{bmatrix}\boldsymbol{u}_{K-1} \\ \boldsymbol{u}_{K-2} \\ \vdots \\ \boldsymbol{u}_1\end{bmatrix} + \boldsymbol{A}^K\boldsymbol{x}_1 = \boldsymbol{0}
\end{aligned}
$$

Controllability (2)

- This is equivalent to a least squares problem:

$$\begin{bmatrix} \boldsymbol{u}_{K-1} \\ \boldsymbol{u}_{K-2} \\ \vdots \\ \boldsymbol{u}_1 \end{bmatrix} = \left[\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{C}^T)^{-1}\right](\boldsymbol{x}_K - \boldsymbol{A}^K\boldsymbol{x}_1)$$

- For $\boldsymbol{C}\boldsymbol{C}^T$ to be invertible, we need $\text{rank}(\boldsymbol{C}) = N$
- $\boldsymbol{C}$ is usually called the **"Controllability Matrix"**
- We follow the same procedure for the time variant case
- In the time invariant case we can stop at N steps (because of the Cayley-Hamilton theorem)

Thank you

- **Any Questions?**
- **Office Hours**:
  - **Tue & Thu** *(09:00-11:00)*
  - 24/7 by email (`costashatz@upatras.gr`, subject: *ECE_RSII_AM*)
- **Material and Announcements**

Laboratory of Automation & Robotics