Robotic Systems II

Lecture 5: Model Predictive Control & Non-linear Trajectory Optimization

Konstantinos Chatzilygeroudis - costashatz@upatras.gr

Department of Electrical and Computer Engineering
University of Patras

Template made by Panagiotis Papagiannopoulos

Bellman's Principle

- Optimal control problems are sequential decision making processes
- Previous control commands affect only the future
- Future controls cannot affect the past
- **Bellman's Principle** says the following:

  "Sub-trajectories of optimal trajectories are optimal for appropriately defined sub-problems."

Dynamic Programming (DP)

- It follows Bellman's Principle directly
- We start from the end/final and move backwards! **Sounds familiar?**

Dynamic Programming (DP)

- It follows Bellman's Principle directly
- We start from the end/final and move backwards! **Sounds familiar?**
- We have seen this in the Riccati recursion
- **Value Function:** $V_k(\boldsymbol{x})$ describes the optimal cost if, at time k, we start at state $\boldsymbol{x}$ and we act optimally there after!
- Very important for Reinforcement Learning! But also control

Dynamic Programming (DP)

- It follows Bellman's Principle directly
- We start from the end/final and move backwards! **Sounds familiar?**
- We have seen this in the Riccati recursion
- **Value Function:** $V_k(\boldsymbol{x})$ describes the optimal cost if, at time k, we start at state $\boldsymbol{x}$ and we act optimally there after!
- Very important for Reinforcement Learning! But also control
- It works only in the discrete case and in specific well-defined optimal control problems

Dynamic Programming Algorithm

Pseudocode:

$$V_K(\boldsymbol{x}) = \ell_F(\boldsymbol{x})$$

$$k = K$$

while $k > 1$

$$V_{k-1}(\boldsymbol{x}) = \min_{\boldsymbol{u}} \left[\ell(\boldsymbol{x}, \boldsymbol{u}) + V_k(f_{\text{discrete}}(\boldsymbol{x}, \boldsymbol{u})) \right]$$

$$k = k - 1$$

Dynamic Programming Algorithm

Pseudocode:

$$V_K(\boldsymbol{x}) = \ell_F(\boldsymbol{x})$$
$$k = K$$
while $k > 1$
$$V_{k-1}(\boldsymbol{x}) = \min_{\boldsymbol{u}} \left[\ell(\boldsymbol{x}, \boldsymbol{u}) + V_k(f_{\text{discrete}}(\boldsymbol{x}, \boldsymbol{u})) \right]$$
$$k = k - 1$$

If we have $V_k(\boldsymbol{x})$, then:

$$\boldsymbol{u}_k(\boldsymbol{x}) = \operatorname{argmin}_{\boldsymbol{u}} \left[\ell(\boldsymbol{x}, \boldsymbol{u}) + V_{k+1}(f_{\text{discrete}}(\boldsymbol{x}, \boldsymbol{u})) \right]$$

In theory, we can solve this online and have the optimal controller! **In practice,** this is usually intractable.

LQR from Dynamic Programming

We start at the end:

$$V_K(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{Q}_K \boldsymbol{x} = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{P}_K \boldsymbol{x}$$

Now we go back:

$$V_{K-1}(\boldsymbol{x}) = \min_{\boldsymbol{u}} \left(\frac{1}{2}\boldsymbol{x}^T \boldsymbol{Q}_{K-1}\boldsymbol{x} + \frac{1}{2}\boldsymbol{u}^T \boldsymbol{R}_{K-1}\boldsymbol{u} + V_K(\boldsymbol{A}_{K-1}\boldsymbol{x} + \boldsymbol{B}_{K-1}\boldsymbol{u}) \right)$$

$$\Rightarrow \boldsymbol{R}_{K-1}\boldsymbol{u} + \boldsymbol{B}_{K-1}^T \boldsymbol{P}_K (\boldsymbol{A}_{K-1}\boldsymbol{x} + \boldsymbol{B}_{K-1}\boldsymbol{u}) = \boldsymbol{0}$$

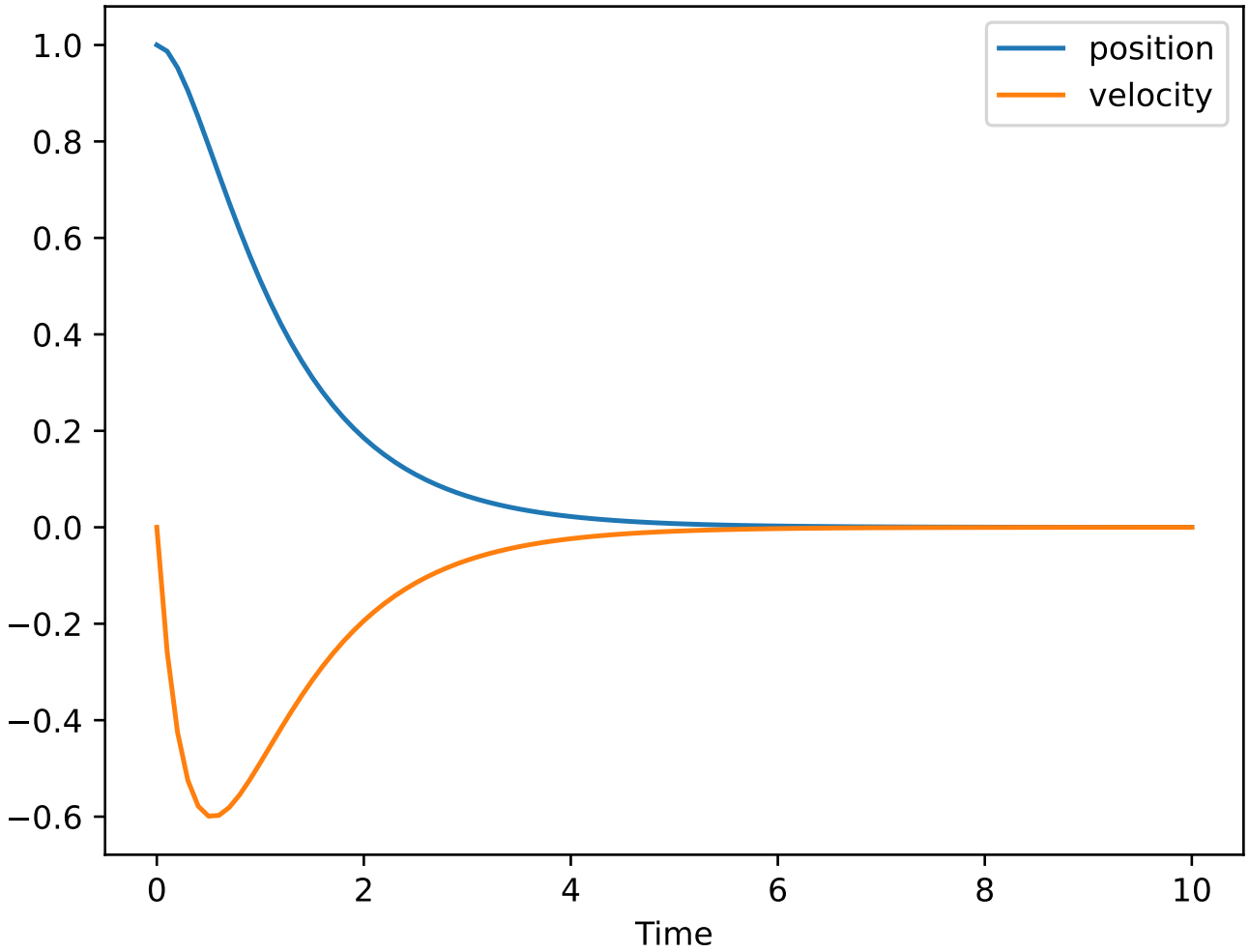
$$\Rightarrow \boldsymbol{u} = -\underbrace{(\boldsymbol{R}_{K-1} + \boldsymbol{B}_{K-1}^T \boldsymbol{P}_K \boldsymbol{B}_{K-1})^{-1} \boldsymbol{B}_{K-1}^T \boldsymbol{P}_K \boldsymbol{A}_{K-1}}_{\boldsymbol{K}_{K-1}} \boldsymbol{x}$$

$$\Rightarrow V_{K-1}(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{P}_{K-1}\boldsymbol{x}$$

where

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

LQR from Dynamic Programming - Code Example



Dynamic Programming - Remarks

- Very important for Reinforcement Learning and Control
- Tractable/useful only for discrete or specific problems (e.g. LQR)
- Even if we assume access to $V(\boldsymbol{x})$, this is in general super non-convex and potentially very hard to optimize (especially online!)
- Curse of dimensionality
- So why did we spend this time on it?

Dynamic Programming - Remarks

- Very important for Reinforcement Learning and Control
- Tractable/useful only for discrete or specific problems (e.g. LQR)
- Even if we assume access to $V(\boldsymbol{x})$, this is in general super non-convex and potentially very hard to optimize (especially online!)
- Curse of dimensionality
- So why did we spend this time on it?
  - Approximate DP: we approximate $V(\boldsymbol{x})$ with a parametric model
  - Foundation of Reinforcement Learning but also useful in Control! We will see in next lectures!

Lagrange Multipliers and Value Function

Let's have a look at the Value function of the LQR problem:

$$V_k(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{P}_k \boldsymbol{x}$$

What if we take the derivative of this:

Lagrange Multipliers and Value Function

Let's have a look at the Value function of the LQR problem:

$$V_k(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{P}_k \boldsymbol{x}$$

What if we take the derivative of this:

$$\nabla_{\boldsymbol{x}} V_k(\boldsymbol{x}) = \boldsymbol{P}_k \boldsymbol{x}$$

What does this reminds us of?

Lagrange Multipliers and Value Function

Let's have a look at the Value function of the LQR problem:

$$V_k(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{P}_k \boldsymbol{x}$$

What if we take the derivative of this:

$$\nabla_{\boldsymbol{x}} V_k(\boldsymbol{x}) = \boldsymbol{P}_k \boldsymbol{x}$$

What does this reminds us of?

$$\nabla_{\boldsymbol{x}} V_k(\boldsymbol{x}) = \boldsymbol{\lambda}_k$$

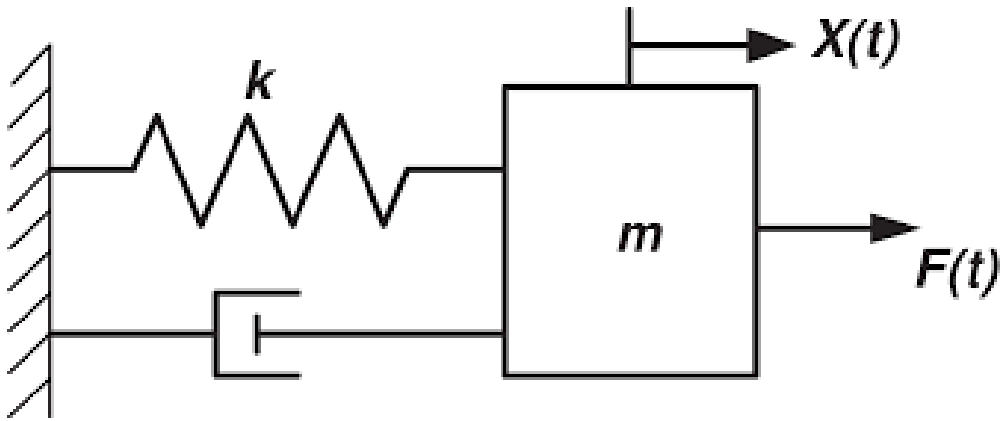
- **The Lagrange multipliers of the dynamics constraints are the gradients of the Value function!!**
- This holds also for the non-linear case! Not just LQR!!

Let's start controlling something!

Mass-Spring-Damper:

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

$$\boldsymbol{u} = m\ddot{\boldsymbol{q}} + c\dot{\boldsymbol{q}} + k\boldsymbol{q}$$



where

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

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

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

$$\boldsymbol{u} = \begin{bmatrix} m\ddot{x} + c\dot{x} + kx \end{bmatrix} \in \mathbb{R}$$

Discrete Dynamics:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ &= \begin{bmatrix} 1 & dt \\ \frac{-kdt}{m} & 1 - \frac{cdt}{m} \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 0 \\ \frac{dt}{m} \end{bmatrix} \boldsymbol{u}_k \end{aligned}$$

Mass-Spring-Damper Control

- **How do we start about controlling this?**

Mass-Spring-Damper Control

- **How do we start about controlling this?**
- Let's run LQR! We get the infinite horizon $\boldsymbol{K}$. Now we have the controller $\boldsymbol{u}(\boldsymbol{x}) = -\boldsymbol{K}\boldsymbol{x}$
- What happens if we have control limits? LQR does not account for that and the Riccati recursion equations fail! What can we do?

Mass-Spring-Damper Control

- **How do we start about controlling this?**
- Let's run LQR! We get the infinite horizon $\boldsymbol{K}$. Now we have the controller $\boldsymbol{u}(\boldsymbol{x}) = -\boldsymbol{K}\boldsymbol{x}$
- What happens if we have control limits? LQR does not account for that and the Riccati recursion equations fail! What can we do?
- **We can use the QP version of LQR!**

Mass-Spring-Damper Control

- **How do we start about controlling this?**
- Let's run LQR! We get the infinite horizon $\boldsymbol{K}$. Now we have the controller $\boldsymbol{u}(\boldsymbol{x}) = -\boldsymbol{K}\boldsymbol{x}$
- What happens if we have control limits? LQR does not account for that and the Riccati recursion equations fail! What can we do?
- **We can use the QP version of LQR!** What is the problem with that?

Mass-Spring-Damper Control

- **How do we start about controlling this?**
- Let's run LQR! We get the infinite horizon $\boldsymbol{K}$. Now we have the controller $\boldsymbol{u}(\boldsymbol{x}) = -\boldsymbol{K}\boldsymbol{x}$
- What happens if we have control limits? LQR does not account for that and the Riccati recursion equations fail! What can we do?
- **We can use the QP version of LQR!** What is the problem with that?
- **QP gives us open-loop trajectories!!** So what can we do?

QP for LQR and Online Control

- Let's remember the original problem:

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

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

QP for LQR and Online Control

- Let's remember the original problem:

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

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

- Here we are solving for the full time horizon K. Can we solve for less steps?

QP for LQR and Online Control

- Let's remember the original problem:

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

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

- Here we are solving for the full time horizon K. Can we solve for less steps?
- Yes, **BUT** is the solution optimal?

QP for LQR and Online Control

- Let's remember the original problem:

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

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

- Here we are solving for the full time horizon K. Can we solve for less steps?
- Yes, **BUT** is the solution optimal? **No!** How can we make it?

QP for LQR and Online Control

- Let's remember the original problem:

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

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

- Here we are solving for the full time horizon K. Can we solve for less steps?
- Yes, **BUT** is the solution optimal? **No!** How can we make it?
- **Idea:** Let's use the Value function: $V_k(\boldsymbol{x}) = \frac{1}{2} \boldsymbol{x}_k^T \boldsymbol{P}_k \boldsymbol{x}_k$

$$\min_{\boldsymbol{x}_{1:H}, \boldsymbol{u}_{1:H-1}} \sum_{k=1}^{H-1} \left(\frac{1}{2} \boldsymbol{x}_k^T \boldsymbol{Q}_k \boldsymbol{x}_k + \frac{1}{2} \boldsymbol{u}_k^T \boldsymbol{R}_k \boldsymbol{u}_k \right) + \frac{1}{2} \boldsymbol{x}_H^T \boldsymbol{P}_H \boldsymbol{x}_H$$

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

- Now we can add as many constraints as we want!

Convex Model Predictive Control

- We refer to such problems as **"Convex Model Predictive Control"**:

$$\min_{\boldsymbol{x}_{1:H}, \boldsymbol{u}_{1:H-1}} \sum_{k=1}^{H-1} \left(\frac{1}{2} \boldsymbol{x}_k^T \boldsymbol{Q}_k \boldsymbol{x}_k + \frac{1}{2} \boldsymbol{u}_k^T \boldsymbol{R}_k \boldsymbol{u}_k \right) + \frac{1}{2} \boldsymbol{x}_H^T \boldsymbol{P}_H \boldsymbol{x}_H$$

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

$$\boldsymbol{u}_k \in \mathcal{U}$$

- Why not do just one step instead of H?

Convex Model Predictive Control

- We refer to such problems as **"Convex Model Predictive Control"**:

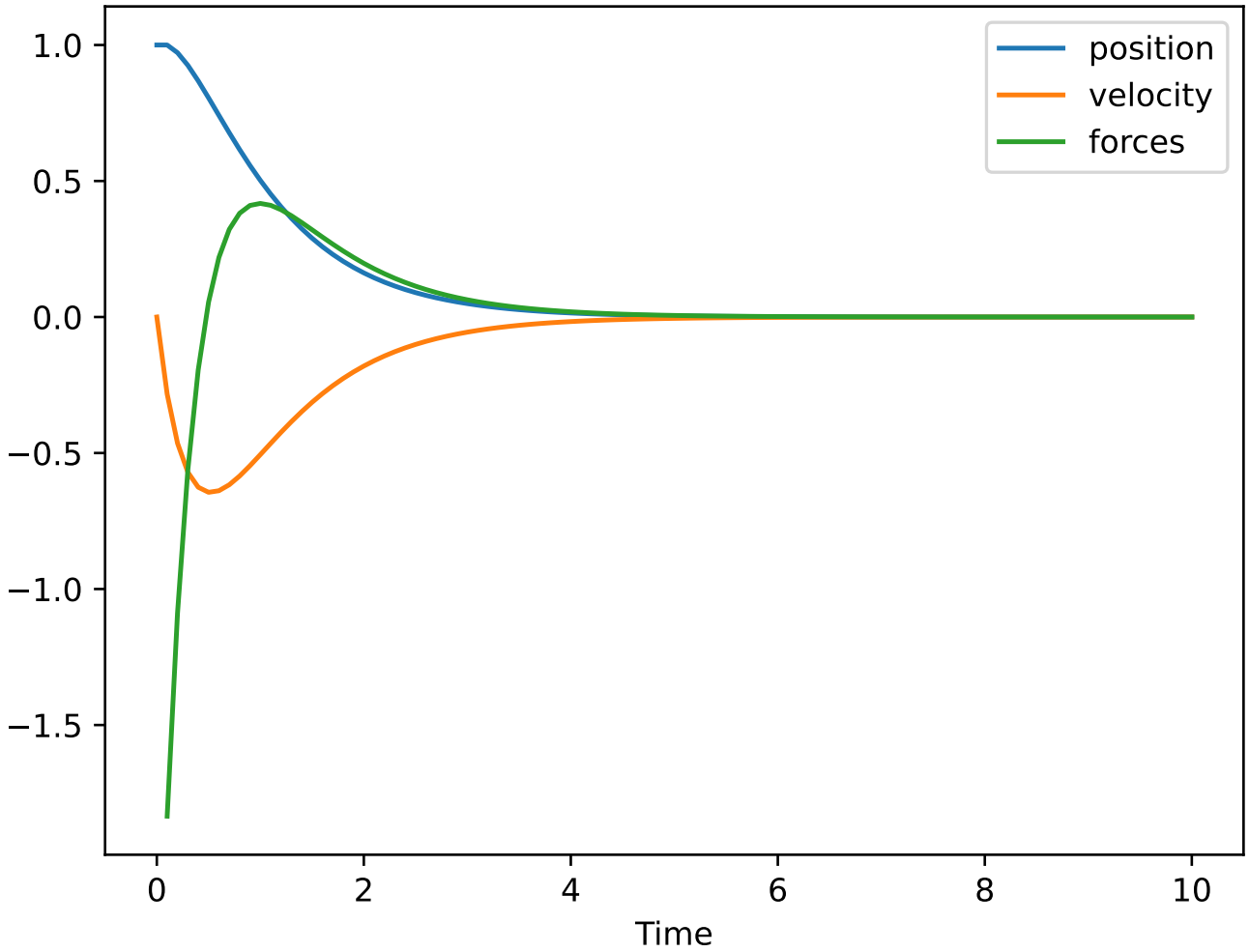
$$\min_{\boldsymbol{x}_{1:H}, \boldsymbol{u}_{1:H-1}} \sum_{k=1}^{H-1} \left(\frac{1}{2} \boldsymbol{x}_k^T \boldsymbol{Q}_k \boldsymbol{x}_k + \frac{1}{2} \boldsymbol{u}_k^T \boldsymbol{R}_k \boldsymbol{u}_k \right) + \frac{1}{2} \boldsymbol{x}_H^T \boldsymbol{P}_H \boldsymbol{x}_H$$

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

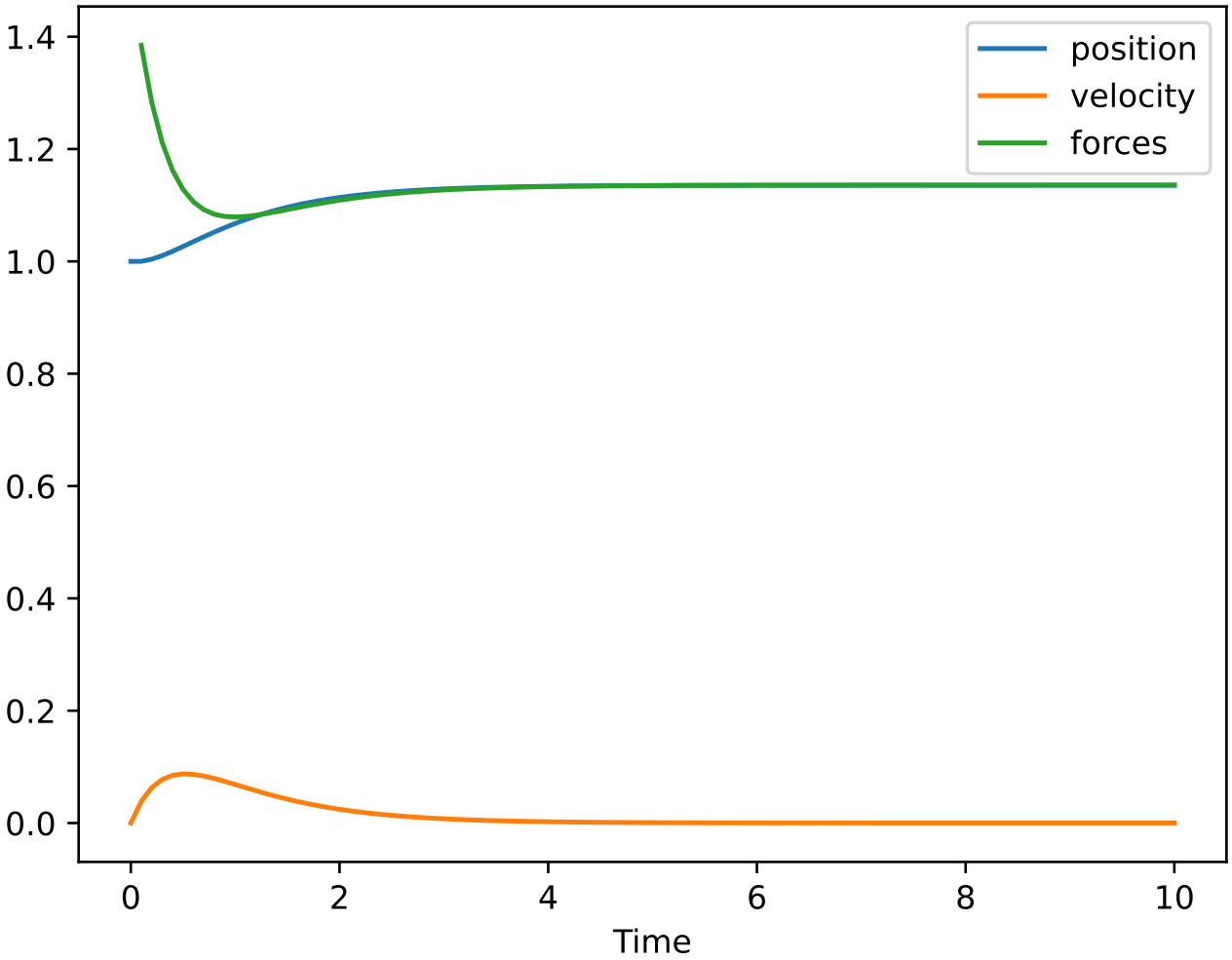
$$\boldsymbol{u}_k \in \mathcal{U}$$

- Why not do just one step instead of H?
- One step is myopic! And $V_k(\boldsymbol{x})$ does not contain information about the new constraints! We might end up in really bad situations!

Convex Model Predictive Control - Code Example



Convex Model Predictive Control - Code Example (2)



LQR for Non-Linear Problems?

- How can we control non linear systems?

LQR for Non-Linear Problems?

- How can we control non linear systems?
- **Idea:** linearize around a fixed point or trajectory and use LQR!
- **What does it mean we "linearize around a fixed point"?**

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

$$\bar{\boldsymbol{x}} = f_{\text{discrete}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \qquad \text{fixed point}$$

LQR for Non-Linear Problems?

- How can we control non linear systems?
- **Idea:** linearize around a fixed point or trajectory and use LQR!
- **What does it mean we "linearize around a fixed point"?**

$$\boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k) \qquad \text{discrete dynamics}$$
$$\bar{\boldsymbol{x}} = f_{\text{discrete}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \qquad \text{fixed point}$$

- Taylor Series around $\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}$:

$$\boldsymbol{x}_{k+1} = f_{\text{discrete}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) + \underbrace{\frac{\partial f_{\text{discrete}}}{\partial \boldsymbol{x}}\Big|_{\bar{\boldsymbol{x}},\bar{\boldsymbol{u}}}}_{\boldsymbol{A}}(\boldsymbol{x}_k - \bar{\boldsymbol{x}}) + \underbrace{\frac{\partial f_{\text{discrete}}}{\partial \boldsymbol{u}}\Big|_{\bar{\boldsymbol{x}},\bar{\boldsymbol{u}}}}_{\boldsymbol{B}}(\boldsymbol{u}_k - \bar{\boldsymbol{u}})$$

We can set $\boldsymbol{x}_k = \bar{\boldsymbol{x}} + \Delta\boldsymbol{x}_k, \boldsymbol{u}_k = \bar{\boldsymbol{u}} + \Delta\boldsymbol{u}_k$ and:

$$\bar{\boldsymbol{x}} + \Delta\boldsymbol{x}_{k+1} = f_{\text{discrete}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) + \boldsymbol{A}\Delta\boldsymbol{x}_k + \boldsymbol{B}\Delta\boldsymbol{u}_k$$
$$\Delta\boldsymbol{x}_{k+1} = \boldsymbol{A}\Delta\boldsymbol{x}_k + \boldsymbol{B}\Delta\boldsymbol{u}_k$$

Convex MPC for Non-Linear Problems?

- Linearize around stable point or trajectory
- Apply LQR to the linearized version
- Convex QP MPC in the linearized version!

Convex MPC for Non-Linear Problems?

- Linearize around stable point or trajectory
- Apply LQR to the linearized version
- Convex QP MPC in the linearized version!
- **Pay attention!** We do the simulation using the non linear dynamics!
- More about this in the lab/homeworks!

Optimal Control for Non-Linear Problems?

Let's remember the problem. Continuous Time:

$$\min_{\boldsymbol{x}(t),\boldsymbol{u}(t)} \mathcal{J}(\boldsymbol{x}(t),\boldsymbol{u}(t)) = \int_{t_0}^{t_f} \ell(\boldsymbol{x}(t),\boldsymbol{u}(t))dt + \ell_F(\boldsymbol{x}(t_f))$$
$$\text{s.t.} \qquad \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t))$$

Discrete Time:

$$\min_{\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}} \mathcal{J}(\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}) = \sum_{k=1}^{K-1} \ell(\boldsymbol{x}_k,\boldsymbol{u}_k) + \ell_F(\boldsymbol{x}_K)$$
$$\text{s.t.} \qquad \boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k,\boldsymbol{u}_k)$$

We transformed $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t))$ to $\boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k,\boldsymbol{u}_k)$.
What did we do here?

Optimal Control for Non-Linear Problems?

Let's remember the problem. Continuous Time:

$$\min_{\boldsymbol{x}(t),\boldsymbol{u}(t)} \mathcal{J}(\boldsymbol{x}(t),\boldsymbol{u}(t)) = \int_{t_0}^{t_f} \ell(\boldsymbol{x}(t),\boldsymbol{u}(t))dt + \ell_F(\boldsymbol{x}(t_f))$$
$$\text{s.t.} \qquad \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t))$$

Discrete Time:

$$\min_{\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}} \mathcal{J}(\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}) = \sum_{k=1}^{K-1} \ell(\boldsymbol{x}_k,\boldsymbol{u}_k) + \ell_F(\boldsymbol{x}_K)$$
$$\text{s.t.} \qquad \boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k,\boldsymbol{u}_k)$$

We transformed $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t))$ to $\boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k,\boldsymbol{u}_k)$. What did we do here? This is **explicit integration!** Can we do implicit? And how?

Non-Linear Trajectory Optimization

- **Key idea:** we discretize but instead of explicit integration, we perform implicit!
- **How?**

Non-Linear Trajectory Optimization

- **Key idea:** we discretize but instead of explicit integration, we perform implicit!
- **How?**
- We have K knot points. There are two main ways of "enforcing the dynamics":
  1. We create splines from the knot points (one spline per 2 knot points) and we enforce the dynamics at the knot points: $\dot{\boldsymbol{x}}_k = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$ (continuous time f) (we can also enforce the dynamics at arbitrary points!)
  2. We perform numerical quadrature
- The first method is generally less accurate but much easier to implement

Cubic Splines

Each pair of sequential knot points $(\boldsymbol{x}_k, \boldsymbol{x}_{k+1})$ define one trajectory with a duration T_k. Assuming $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}$, we model the trajectory $\boldsymbol{q}_k(t)$ as follows:

$$\boldsymbol{q}_k(t) = \boldsymbol{c}_3 t^3 + \boldsymbol{c}_2 t^2 + \boldsymbol{c}_1 t + \boldsymbol{c}_0$$

Important to note: $\boldsymbol{q}_k(t)$ defines the trajectory that starts from $\boldsymbol{x}_k$ and ends to $\boldsymbol{x}_{k+1}$ with duration T_k. Assuming that $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ are the optimization variables, we can construct the $\boldsymbol{c}$'s as follows:

$$\begin{aligned}
\boldsymbol{c}_0 &= \boldsymbol{q}_k \\
\boldsymbol{c}_1 &= \dot{\boldsymbol{q}}_k \\
\boldsymbol{c}_2 &= \frac{3\boldsymbol{q}_{k+1}}{T_k^2} - \frac{3\boldsymbol{q}_k}{T_k^2} - \frac{2\dot{\boldsymbol{q}}_k}{T_k} - \frac{\dot{\boldsymbol{q}}_{k+1}}{T_k} \\
\boldsymbol{c}_3 &= -\frac{2\boldsymbol{q}_{k+1}}{T_k^3} + \frac{2\boldsymbol{q}_k}{T_k^3} + \frac{\dot{\boldsymbol{q}}_k}{T_k^2} + \frac{\dot{\boldsymbol{q}}_{k+1}}{T_k^2}
\end{aligned}$$

Cubic Splines (2)

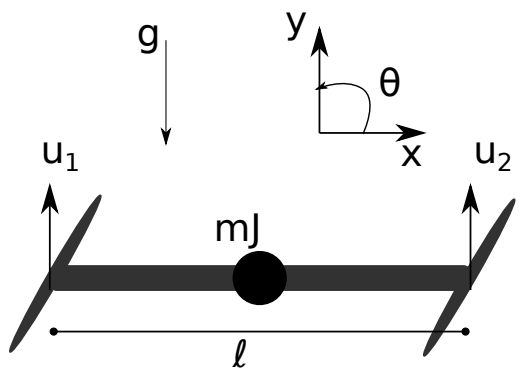
- We have now defined $\boldsymbol{x}$ as a function of $\boldsymbol{x}_{1:K}$, i.e.: $\boldsymbol{x}(\boldsymbol{x}_{1:K})$
- We can still query it at any time: $\boldsymbol{x}(t, \boldsymbol{x}_{1:K}) = \begin{bmatrix} \boldsymbol{q}(t, \boldsymbol{x}_{1:K}) \\ \dot{\boldsymbol{q}}(t, \boldsymbol{x}_{1:K}) \end{bmatrix}$
- We can do similar things for controls $\boldsymbol{u}(t)$ or even simpler (e.g. zero order hold)
- We can also easily compute all gradients, e.g. $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}_k}$. We can even compute $\frac{\partial \boldsymbol{x}}{\partial T_k}$: this means that we can optimize for the duration of each part!

Planar Quadrotor Control

Planar Quadrotor:

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

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$



where

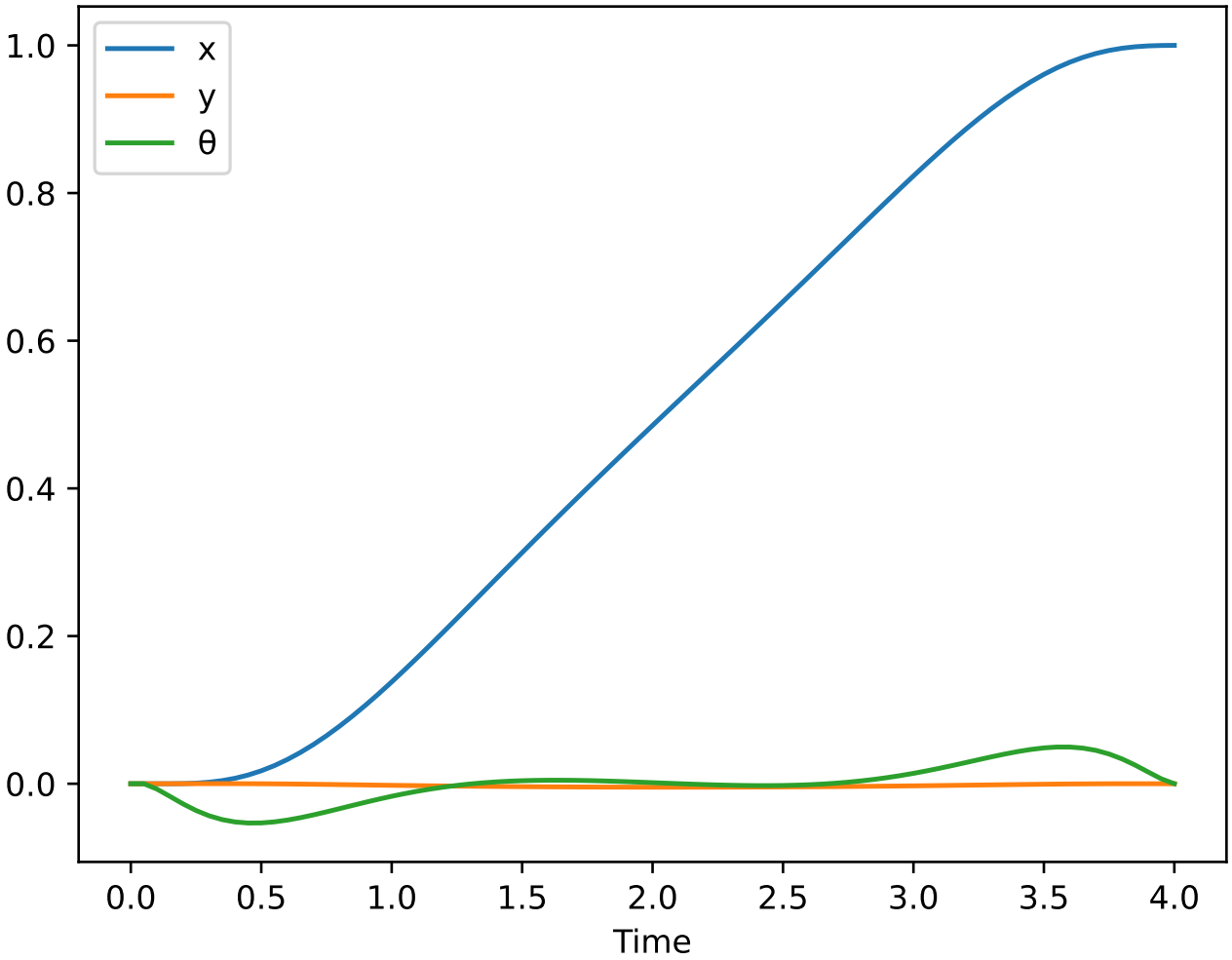
$$\boldsymbol{q} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \in \mathbb{R}^3$$

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \in \mathbb{R}^3$$

Dynamics:

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{-(u_1+u_2)\sin\theta}{m} \\ \frac{(u_1+u_2)\cos\theta}{m} - g \\ \frac{l(u_2-u_1)}{2J} \end{bmatrix}$$

Non-Linear Trajectory Optimization - Code Example



Quadrature - Trapezoidal Rule

The **trapezoidal rule** is a technique for approximating definite integrals. For example, we have:

$$\int_{t_1}^{t_K} f(t)dt \approx \frac{1}{2}(t_K - t_1)\big(f(t_K) + f(t_1)\big)$$

We can expand this and take many smaller steps:

$$\int_{t_1}^{t_K} f(t)dt \approx \sum_{k=1}^{K-1} \frac{\big(f(t_{k+1}) + f(t_k)\big)}{2}(t_{k+1} - t_k)$$

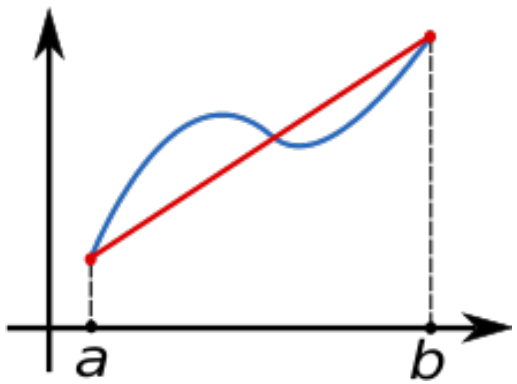


Figure: Source: Wikipedia

Trajectory Optimization with the Trapezoidal Rule

- **Cost function:**

$$\int_{t_0}^{t_f} \ell(\boldsymbol{x}(t), \boldsymbol{u}(t))dt \approx \sum_{k=1}^{K-1} \frac{(\ell_{k+1} + \ell_k)}{2}(t_{k+1} - t_k)$$

  where $\ell_k = \ell(\boldsymbol{x}_k, \boldsymbol{u}_k)$.

- **Constraints (apart from dynamics):** We evaluate them at the "knot points" only!

- **Dynamics Constraints** (for each segment):

$$\int_{t_k}^{t_{k+1}} \dot{\boldsymbol{x}}(t)dt = \int_{t_k}^{t_{k+1}} f(\boldsymbol{x}(t), \boldsymbol{u}(t))dt$$

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k \approx \frac{1}{2}(t_{k+1} - t_k)\big(f_{k+1} + f_k\big)$$

  where $f_k = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$

Trajectory Optimization with the Trapezoidal Rule (2)

- We optimized. Now how can we get trajectories back?

Trajectory Optimization with the Trapezoidal Rule (2)

- We optimized. Now how can we get trajectories back?
- **Interpolation** (for each segment):
  - **Controls:**

$$\boldsymbol{u}(t) \approx \boldsymbol{u}_k + \frac{t - t_k}{t_{k+1} - t_k}(\boldsymbol{u}_{k+1} - \boldsymbol{u}_k)$$

  - **State:**

$$\dot{\boldsymbol{x}}(t) \approx f_k + \frac{t - t_k}{t_{k+1} - t_k}(f_{k+1} - f_k)$$

$$\boldsymbol{x}(t) = \int \dot{\boldsymbol{x}}(t)dt \approx \boldsymbol{x}_k + f_k(t - t_k) + \frac{(t - t_k)^2}{2(t_{k+1} - t_k)}(f_{k+1} - f_k)$$

TO with the Hermite–Simpson Rule

Just like the trapezoidal rule, but we have the following rule:

$$\int_{t_1}^{t_K} f(t)dt \approx \frac{(t_K - t_1)}{6}\Big(f(t_1) + 4f\big(\frac{t_1 + t_K}{2}\big) + f(t_K)\Big)$$

For the dynamics constraints we end up with:

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k \approx \frac{1}{6}(t_{k+1} - t_k)(f_k + 4f_{k+\frac{1}{2}} + f_{k+1}) \quad (1)$$

$$\boldsymbol{x}_{k+\frac{1}{2}} \approx \frac{1}{2}(\boldsymbol{x}_{k+1} + \boldsymbol{x}_k) + \frac{t_{k+1} - t_k}{8}(f_k - f_{k+1}) \quad (2)$$

We can either combine equations (1) and (2) into one constraint or add $\boldsymbol{x}_{k+\frac{1}{2}}$ as optimization variables. We evaluate at knot and mid points the rest of the constraints as well.

TO with the Hermite–Simpson Rule (2)

For interpolation we get (for each segment):

- **Controls:**

$$\begin{aligned}\boldsymbol{u}(t) \approx & \frac{2}{(t_{k+1}-t_k)^2}(t-t_k-\frac{(t_{k+1}-t_k)}{2})(t-t_{k+1})\boldsymbol{u}_k \\ & -\frac{4}{(t_{k+1}-t_k)^2}(t-t_k)(t-t_{k+1})\boldsymbol{u}_{k+\frac{1}{2}} \\ & +\frac{2}{(t_{k+1}-t_k)^2}(t-t_k)(t-t_k-\frac{(t_{k+1}-t_k)}{2})\boldsymbol{u}_{k+1}\end{aligned}$$

- **State:**

$$\begin{aligned}\boldsymbol{x}(t) \approx & \boldsymbol{x}_k + f_k(\frac{t-t_k}{t_{k+1}-t_k}) + \frac{1}{2}(-3f_k + 4f_{k+\frac{1}{2}} - f_{k+1})(\frac{t-t_k}{t_{k+1}-t_k})^2 \\ & + \frac{1}{3}(2f_k - 4f_{k+\frac{1}{2}} + 2f_{k+1})(\frac{t-t_k}{t_{k+1}-t_k})^3\end{aligned}$$

TO with the Hermite–Simpson Rule (3)

- Hermite–Simpson rule more accurate than the trapezoidal rule. Why?

TO with the Hermite–Simpson Rule (3)

- Hermite–Simpson rule more accurate than the trapezoidal rule. Why?
  - Trapezoidal produces piecewise linear actions and piecewise quadratic states
  - Hermite–Simpson produces piecewise quadratic actions and piecewise cubic states
- Hermite–Simpson rule can be written via cubic splines
- We refer to those methods as **"Direct Collocation Methods"**:
  - The middle points $\boldsymbol{x}_{k+\frac{1}{2}}, \boldsymbol{u}_{k+\frac{1}{2}}$ are called **collocation points**!

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