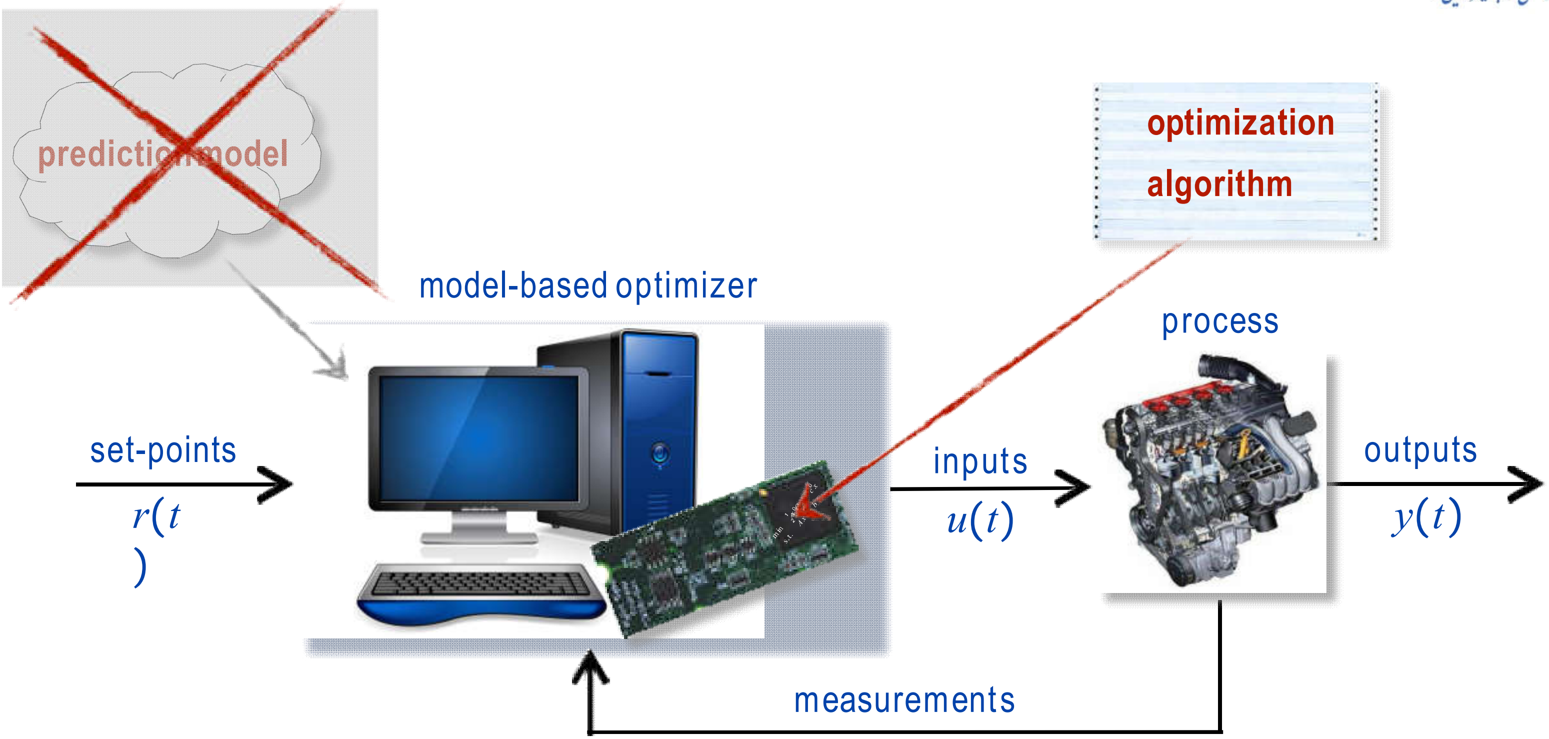
کنترل پیش بین
Model Predictive Control

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

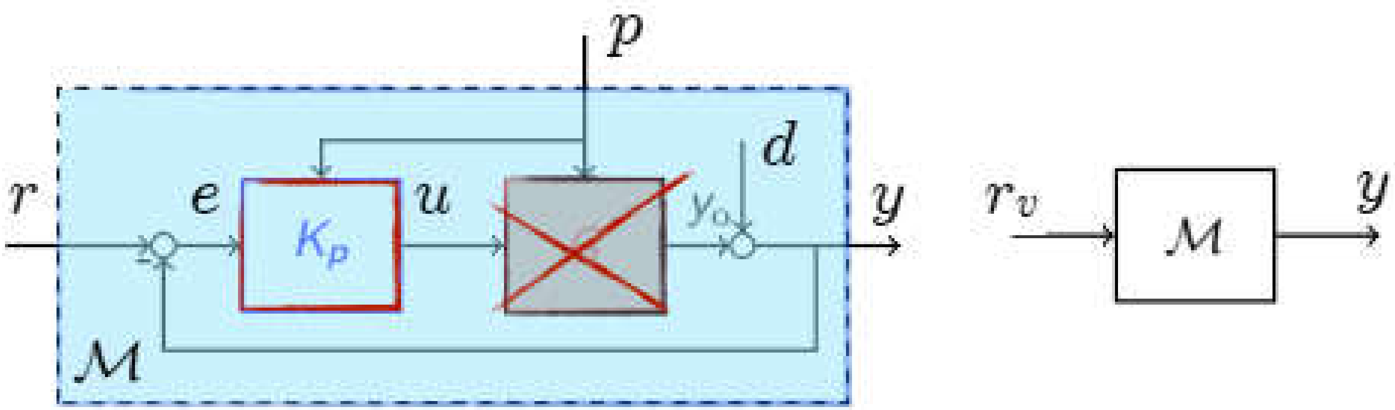
Data-driven MPC



- Can we design an MPC controller **without** first identifying a model of the **open-loop process**?

Data-driven direct controller synthesis

(Campi, Lecchini, Savaresi, 2002) (Formentin et al., 2015)

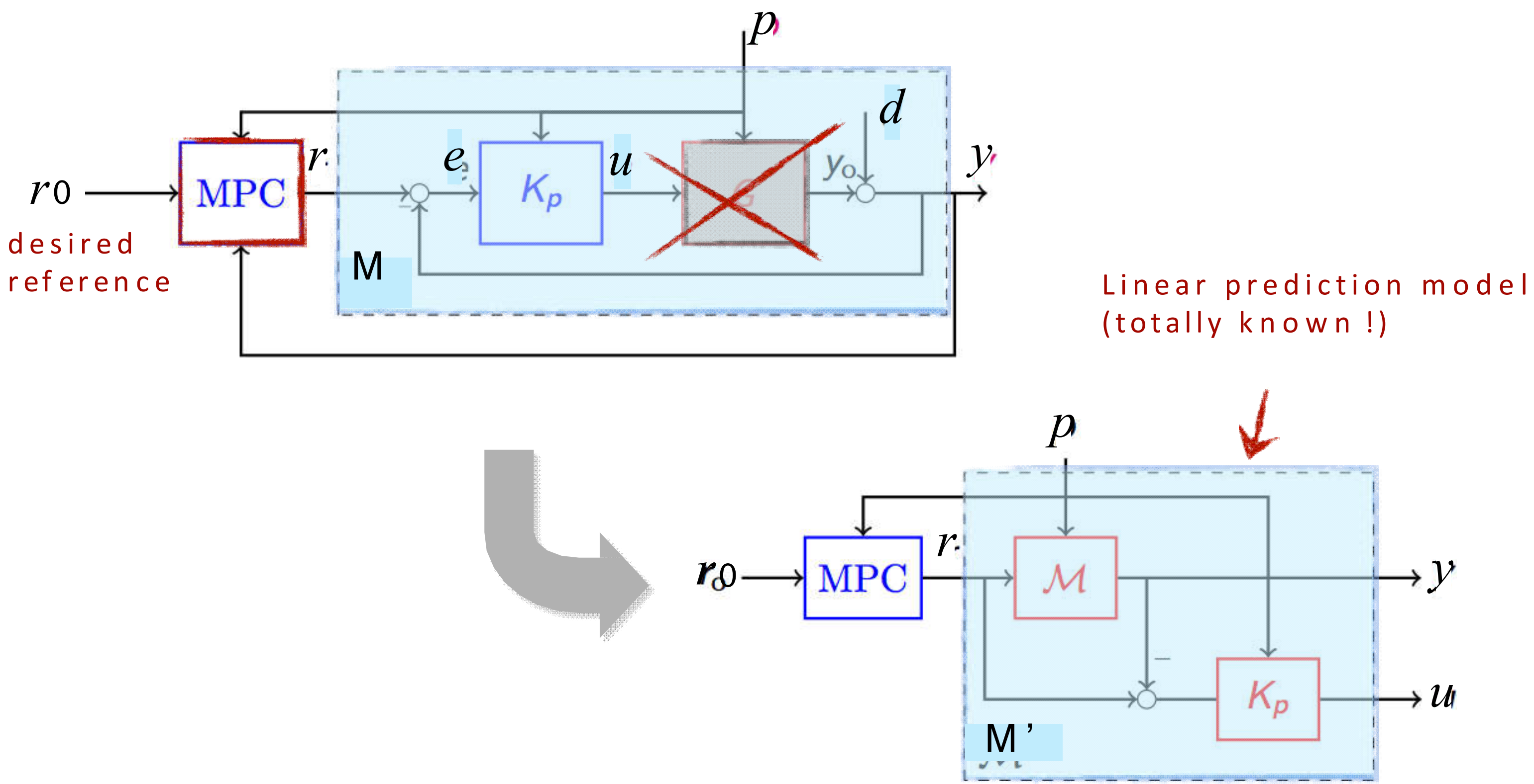


- Collect a set of **data** $\{u(t), y(t), p(t)\}, t = 1, \ldots, N$
- Specify a **desired closed-loop linear model** $\mathcal{M}$ from r to y
- Compute $r_v(t) = \mathcal{M}^{\#} y(t)$ from **pseudo-inverse model** $\mathcal{M}^{\#}$ of $\mathcal{M}$
- **Identify** linear (LPV) model K_p from $e_v = r_v - y$ (virtual tracking error) to u

Data-driven MPC

- Design a linear MPC (**reference governor**) to generate the reference r

  (Bemporad, Mosca, 1994) (Gilbert, Kolmanovsky, Tan, 1994)



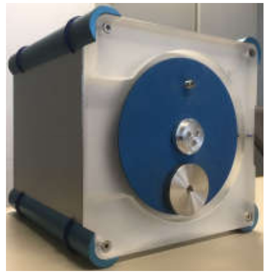
- MPC designed to handle input/output **constraints** and improve **performance**

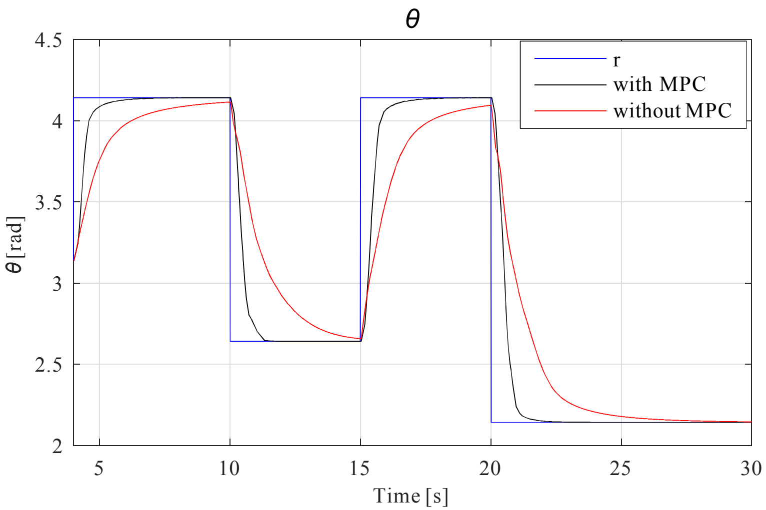
  (Piga, Formentin, Bemporad, 2017)

Data-driven MPC - An example

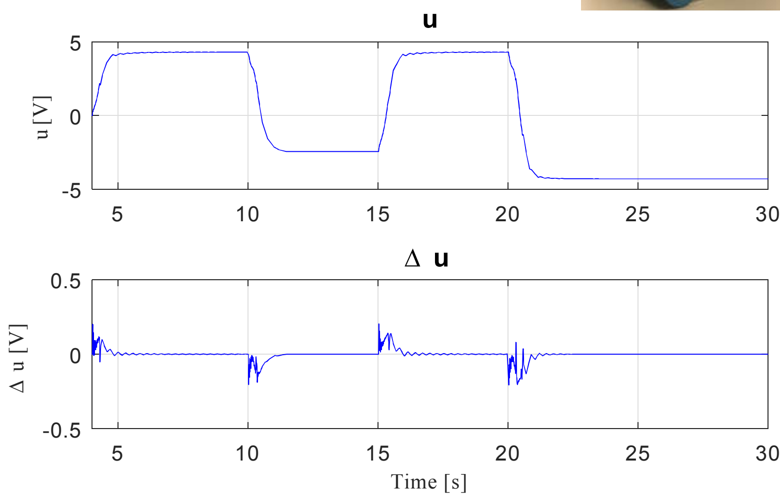
- Experimental results: MPC handles soft constraints on u, Δu and y

(motor equipment by courtesy of TU Delft)





desired tracking performance achieved

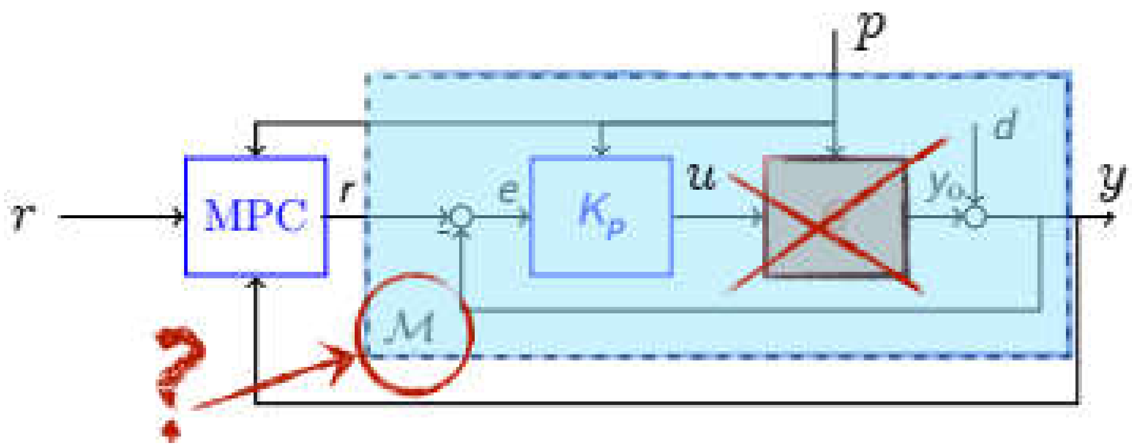


constraints on input increments satisfied

No open-loop process model is identified to design the MPC controller!

Optimal data-driven MPC

- **Question**: How to choose the reference model $\mathcal{M}$?



- Can we choose $\mathcal{M}$ from data so that K_p is an **optimal controller** ?

Optimal data-driven MPC

- **Idea**: parameterize desired closed-loop model $\mathcal{M}(\theta)$ and optimize

$$\min_{\theta} J(\theta) = \frac{1}{N} \sum_{t=0}^{N-1} \underbrace{W_y(r(t) - y_p(\theta, t))^2 + W_{\Delta u} \Delta u_p^2(\theta, t)}_{\text{performance index}} + \underbrace{W_{\text{fit}}(u(t) - u_v(\theta, t))^2}_{\text{identification error}}$$

- Evaluating $J(\theta)$ requires synthesizing $K_p(\theta)$ from data and simulating the nominal model and control law

$$y_p(\theta, t) = \mathcal{M}(\theta) r(t) \qquad u_p(\theta, t) = K_p(\theta)(r(t) - y_p(\theta, t))$$

$$\Delta u_p(\theta, t) = u_p(\theta, t) - u_p(\theta, t-1)$$

- Optimal θ obtained by solving a (non-convex) nonlinear programming problem

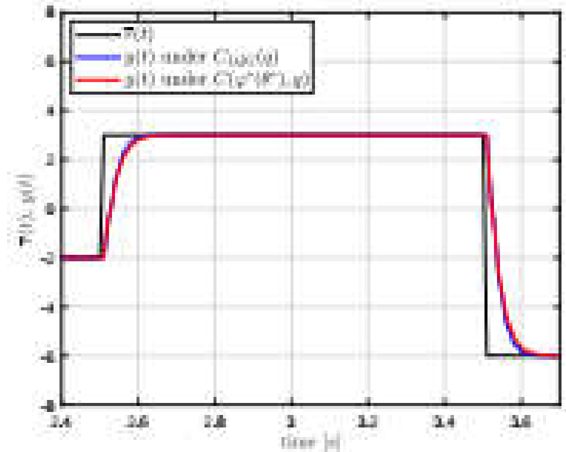
Optimal data-driven MPC

(Selvi, Piga, Bemporad, 2018)

- **Results:** linear process

$$G(z) = \frac{z - 0.4}{z^2 + 0.15z - 0.325}$$

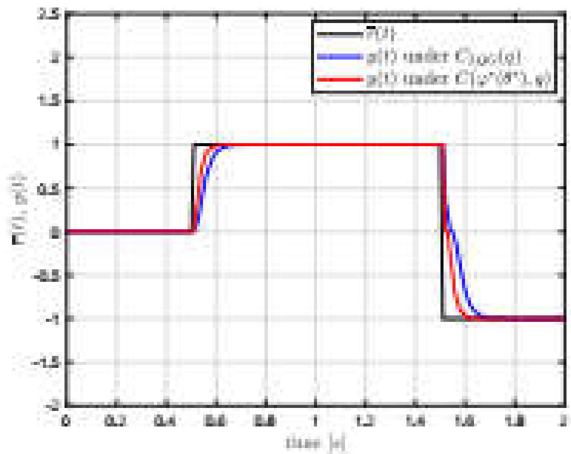
Data-driven controller only 1.3% worse than model-based LQR (=SYS-ID on same data + LQR design)



- **Results:** nonlinear (Wiener) process

$$\begin{aligned} y_L(t) &= G(z)u(t) \\ y(t) &= |y_L(t)| \arctan(y_L(t)) \end{aligned}$$

The data-driven controller is 24% better than LQR based on identified open-loop model !



Data-driven optimal policy search

- Plant + environment dynamics (unknown):

$$s_{t+1} = h(s_t, p_t, u_t, d_t)$$

  - s_t states of plant & environment
  - p_t exogenous signal (e.g., reference)
  - u_t control input
  - d_t unmeasured disturbances

- Control policy: $\pi : \mathbb{R}^{n_s+n_p} \longrightarrow \mathbb{R}^{n_u}$ deterministic control policy

$$u_t = \pi(s_t, p_t)$$

- Closed-loop performance of an execution is defined as

$$\mathcal{J}_\infty(\pi, s_0, \{p_\ell, d_\ell\}_{\ell=0}^{\infty}) = \sum_{\ell=0}^{\infty} \rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell))$$

$$\rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell)) = \text{stage cost}$$

Optimal Policy Search Problem

- **Optimal policy:**

$$\pi^* = \arg\min_\pi \mathcal{J}(\pi)$$
$$\mathcal{J}(\pi) = \mathbb{E}_{s_0,\{p_\ell,d_\ell\}}\left[\mathcal{J}_\infty(\pi, s_0, \{p_\ell, d_\ell\})\right]$$

expected performance

- **Simplifications:**

  - Finite parameterization: $\pi = \pi_K(s_t, p_t)$ with K = parameters to optimize
  - Finite horizon: $\mathcal{J}_L(\pi, s_0, \{p_\ell, d_\ell\}_{\ell=0}^{L-1}) = \sum_{\ell=0}^{L-1} \rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell))$

- Optimal policy search: use **stochastic gradient descent (SGD)**

$$K_t \leftarrow K_{t-1} - \alpha_t \mathcal{D}(K_{t-1})$$

with $\mathcal{D}(K_{t-1})$ = descent direction

Descent Direction

- The descent direction $\mathcal{D}(K_{t-1})$ is computed by generating:
  - N_s perturbations $s_0^{(i)}$ around the current state s_t
  - N_r random reference signals $r_\ell^{(j)}$ of length L,
  - N_d random disturbance signals $d_\ell^{(h)}$ of length L,

$$\mathcal{D}(K_{t-1}) = \sum_{i=1}^{N_s} \sum_{j=1}^{N_p} \sum_{k=1}^{N_q} \nabla_K \mathcal{J}_L(\pi_{K_{t-1}}, s_0^{(i)}, \{r_\ell^{(j)}, d_\ell^{(k)}\})$$

SGD step = mini-batch of size $M = N_s \cdot N_r \cdot N_d$

- Computing $\nabla_K \mathcal{J}_L$ requires predicting the effect of π over L future steps
- We use a local linear model just for computing $\nabla_K \mathcal{J}_L$, obtained by running **recursive linear system identification**

Optimal Policy Search Algorithm

- At each step t:
  1. Acquire current s_t
  2. Recursively update the local linear model
  3. Estimate the direction of descent $\mathcal{D}(K_{t-1})$
  4. Update policy: $K_t \leftarrow K_{t-1} - \alpha_t \mathcal{D}(K_{t-1})$

- If policy is learned online and needs to be applied to the process:
  - Compute the nearest policy K_t^* to K_t that stabilizes the local model

$$K_t^* = \underset{K}{\text{argmin}} \|K - K_t^s\|_2^2$$
$$\text{s.t. } K \text{ stabilizes local linear model}$$

linear matrix inequality

- When policy is learned online, exploration is guaranteed by the reference r_t

Special Case: Output Tracking

- $x_t = [y_t, y_{t-1}, \ldots, y_{t-n_o}, u_{t-1}, u_{t-2}, \ldots, u_{t-n_i}]$

  $\Delta u_t = u_t - u_{t-1}$ control input increment

- Stage cost: $\| y_{t+1} - r_t \|_{Qy}^2 + \| \Delta u_t \|_R^2 + \| q_{t+1} \|_{Q_q}^2$

- Integral action dynamics $q_{t+1} = q_t + (y_{t+1} - r_t)$

$$\Rightarrow \quad s_t = \begin{bmatrix} x_t \\ q_t \end{bmatrix}, \quad p_t = r_t.$$

- **Linear policy parametrization:**

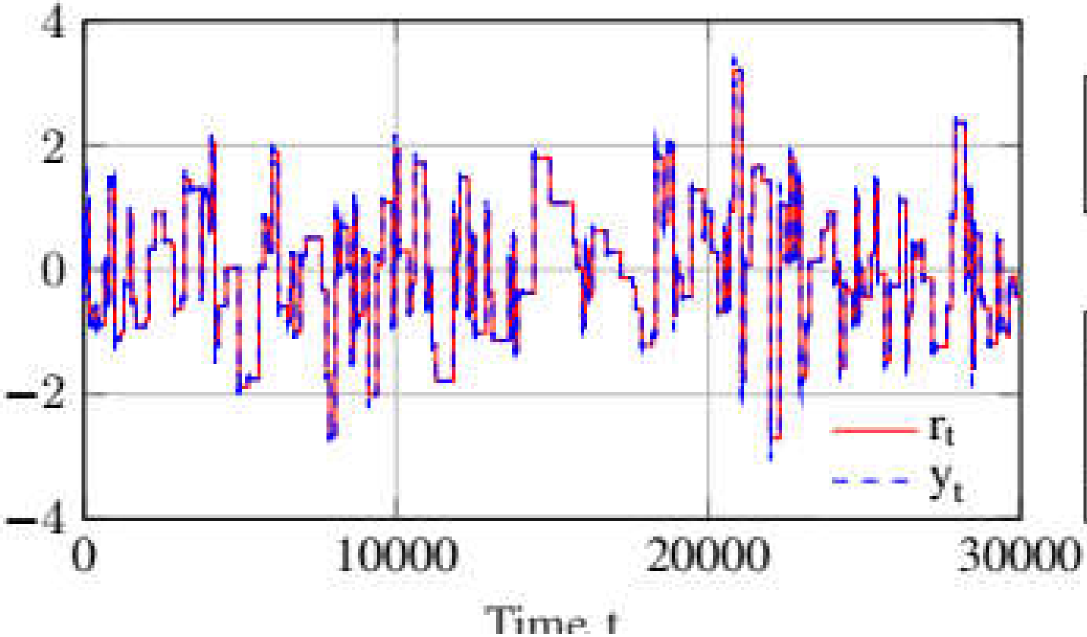
$$\pi_K(s_t, r_t) = -K^s \cdot s_t - K^r \cdot r_t, \qquad K = \begin{bmatrix} K^s \\ K^r \end{bmatrix}$$

Special Case: Output Tracking

$$\begin{cases} x_{t+1} = \begin{bmatrix} -0.669 & 0.378 & 0.233 \\ -0.288 & -0.147 & -0.638 \\ -0.337 & 0.589 & 0.043 \end{bmatrix} x_t + \begin{bmatrix} -0.295 \\ -0.325 \\ -0.258 \end{bmatrix} u_t \\ y_t = \begin{bmatrix} -1.139 & 0.319 & -0.571 \end{bmatrix} x_t \end{cases}$$

model is unknown

Online tracking performance (no disturbance, $d_t = 0$):



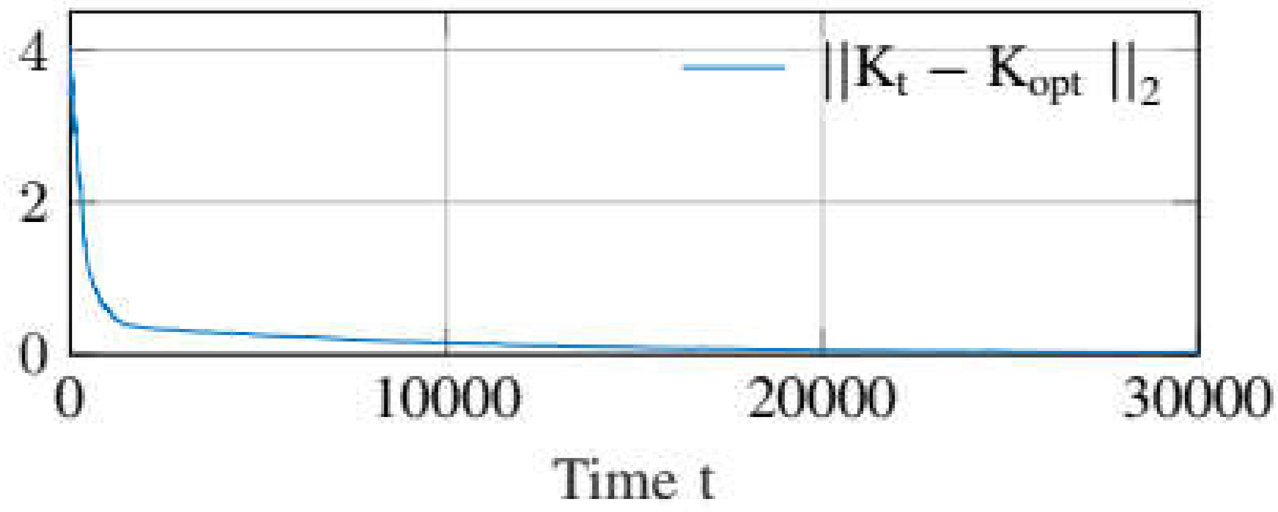
$Q_y = 1$
$R = 0.1$
$Q_q = 1$

n_i	n_o	L
3	3	20

N_0	N_r	N_q
50	1	10

Special Case: Output Tracking

Evolution of the error $\|K_t - K_{opt}\|_2$:



$$K_{\mathrm{SGD}} = [-1.255, 0.218, 0.652, 0.895, 0.050, 1.115, -2.186]$$

$$K_{\mathrm{opt}} = [-1.257, 0.219, 0.653, 0.898, 0.050, 1.141, -2.196]$$

Learning MPC from data

- **Goal:** learn **MPC law** from data that optimizes a given
- **Reinforcement learning** = use **data** and a **performance index** to learn an optimal policy
- **Q-learning**: learn Q-function defining the MPC law from data

  (Gros, Zanon, 2019) (Zanon, Gros, Bemporad, 2019)
- **Policy gradient methods**: learn optimal policy coefficients directly from data using stochastic gradient descent (Ferrarotti, Bemporad, 2019)
- **Global optimization methods**: learn MPC parameters (weights, models, horizon, solver tolerances, ...) by optimizing observed closed-loop performance (Piga, Forgione, Formentin, Bemporad, 2019) (Forgione, Piga, Bemporad, 2020)

Learning MPC from data

Model/policy structure **includes** real plant/optimal policy:
– **Sys-id + model-based** synthesis = model-free **reinforcement learning**
– Reinforcement learning **may** require more data (model-based can instead "extrapolate" optimal actions)

Model/policy structure **does not include** real plant/optimal policy:
– optimal policy **learned from data** **may** be better than **model-based** optimal policy
– when open-loop model is used as a tuning parameter, **learned model** can be quite different from best **open-loop model** that can be identified from the same data