MODEL PREDICTIVE CONTROL

DATA-DRIVEN MPC

Alberto Bemporad

imt.lu/ab

COURSE STRUCTURE

- ✔ **Basic concepts** of model predictive control (MPC) and **linear MPC**
- ✔ **Linear time-varying** and **nonlinear** MPC
- ✔ MPC computations: **quadratic programming** (QP), **explicit** MPC
- ✔ Hybrid MPC
- ✔ Stochastic MPC
- Data-driven MPC

Course page:

`http://cse.lab.imtlucca.it/~bemporad/mpc_course.html`

LEARNING MPC FROM DATA

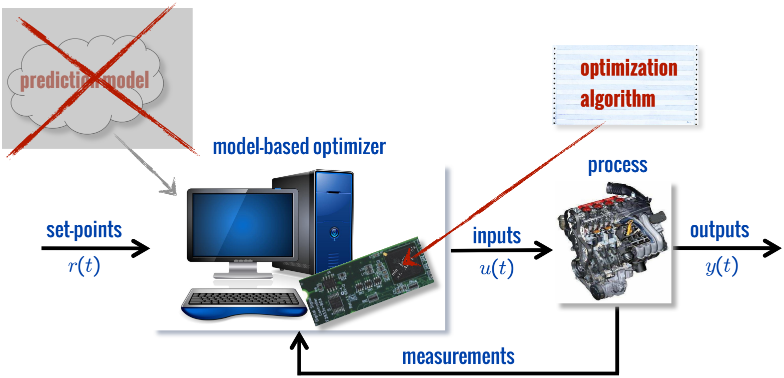
- **Goal**: learn MPC law from data that optimizes a given index
- **Reinforcement learning** = use **data** and a **performance index** to learn an optimal policy
- **Q-learning**: learn Q-function defining the MPC law from data

  (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)

DATA-DRIVEN MPC

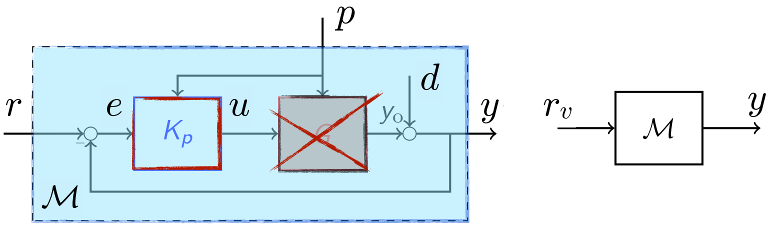
DATA-DRIVEN MPC



- 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)

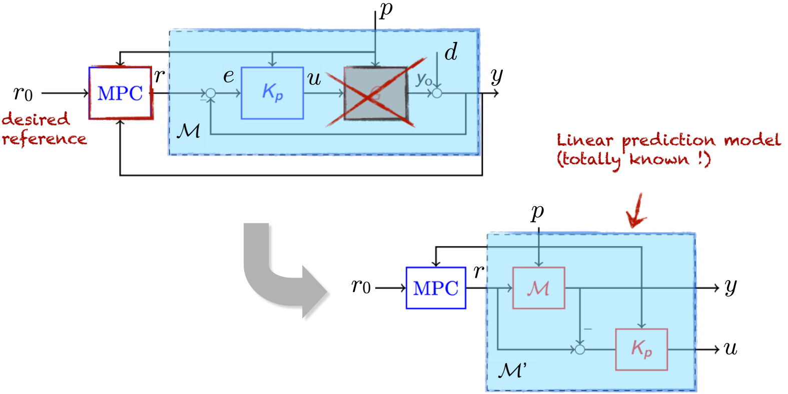


- Collect a set of **data** $\{u(t), y(t), p(t)\}, t = 1, \ldots, N$
- Specify a **desired closed-loop linear model** $\mathcal{M}$ from r to y
- Compute $r_v(t) = \mathcal{M}^{\#} y(t)$ from **pseudo-inverse model** $\mathcal{M}^{\#}$ of $\mathcal{M}$
- **Identify** linear (LPV) model K_p from $e_v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



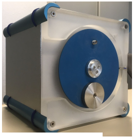
- MPC designed to handle input/output **constraints** and improve **performance**

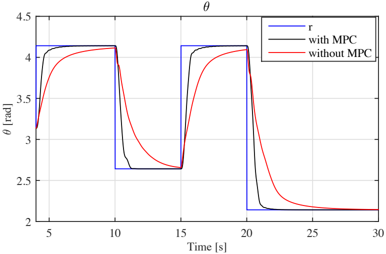
  (Piga, Formentin, Bemporad, 2017)

DATA-DRIVEN MPC - AN EXAMPLE

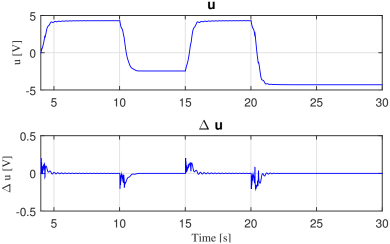
- Experimental results: MPC handles soft constraints on u, Δu and y

  (motor equipment by courtesy of TU Delft)





desired tracking performance achieved

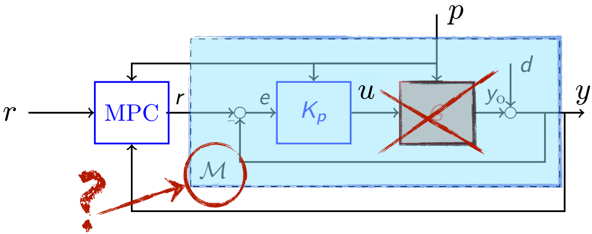


constraints on input increments satisfied

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

(Selvi, Piga, Bemporad, 2018)

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

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

- 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

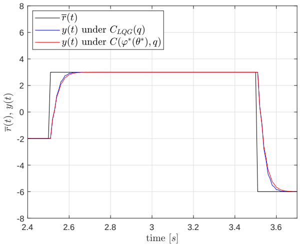
OPTIMAL DATA-DRIVEN MPC

(Selvi, Piga, Bemporad, 2018)

- **Results**: linear process

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

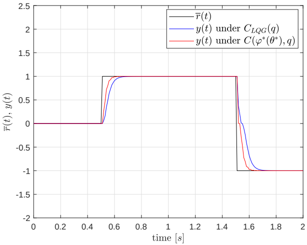
Data-driven controller **only 1.3% worse** than model-based LQR (=SYS-ID on same data + LQR design)



- **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

DATA-DRIVEN OPTIMAL POLICY SEARCH

(Ferrarotti, Bemporad, 2019)

- 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

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

$\rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell)) \;=\;$ stage cost

OPTIMAL POLICY SEARCH PROBLEM

- **Optimal policy:**

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

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)**

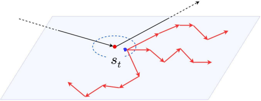
$$K_t \leftarrow K_{t-1} - \alpha_t \mathcal{D}(K_{t-1})$$

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 $K_t^\star$ to K_t that stabilizes the local model

$$\begin{aligned} K_t^\star = \quad & \text{arg}\min_K \|K - K_t^s\|_2^2 \\ & \text{s.t. } K \text{ stabilizes local linear model} \end{aligned}$$

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

- 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:**

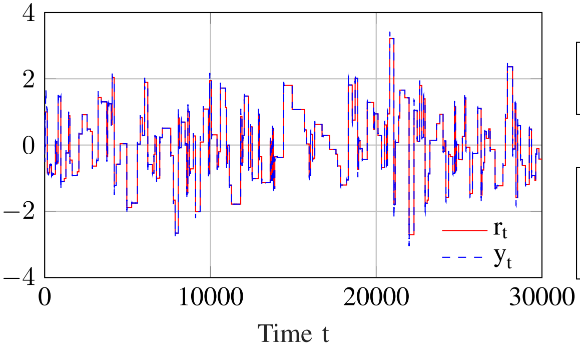
$$\pi_K(s_t,\, r_t) = -K^s \cdot s_t - K^r \cdot r_t, \qquad K = \begin{bmatrix} K^s \\ K^r \end{bmatrix}$$

EXAMPLE: RETRIEVE LQR FROM DATA

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

model is unknown

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



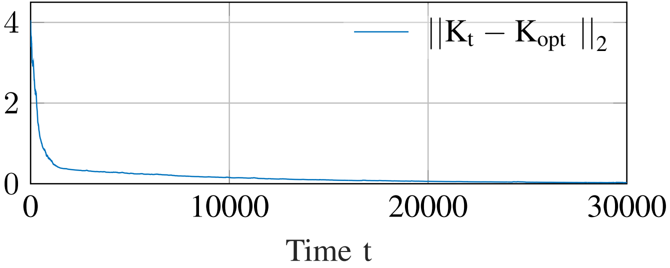
$Q_y = 1$
$R = 0.1$
$Q_q = 1$

n_i	n_o	L
3	3	20

N_0	N_r	N_q
50	1	10

EXAMPLE: RETRIEVE LQR FROM DATA

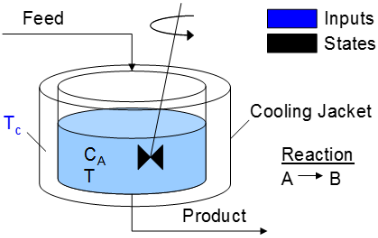
Evolution of the error $\|K_t - K_{opt}\|_2$:



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

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

NONLINEAR EXAMPLE



Continuously Stirred Tank Reactor (CSTR)[1]

model is unknown

Feed:

- concentration: 10kg mol/m^3
- temperature: 298.15K

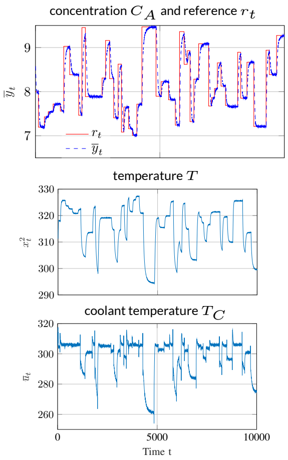
$$T = \hat{T} + \eta_T,\; C_A = \hat{C_A} + \eta_C, \quad \eta_T, \eta_C \sim \mathcal{N}(0, \sigma^2), \quad \sigma = 0.01$$

$$\boxed{Q_y = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad R = 0.1 \qquad Q_q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0 \end{bmatrix}}$$

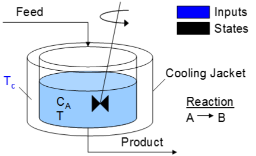
[1] figure retrived from apmonitor.com

NONLINEAR EXAMPLE

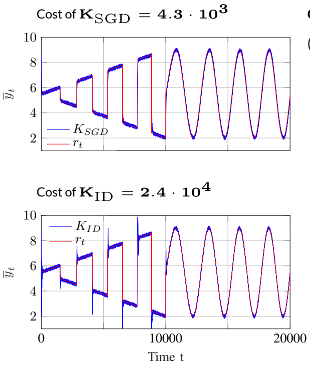
Online learning



n_i	n_o	L
2	3	10
N_0	N_r	N_q
50	20	20



Validation phase



Continuously Stirred Tank Reactor (CSTR)

(courtesy: apmonitor.com)

SGD beats SYS-ID + LQR

- Approach currently extended to multiple-linear and nonlinear policies

AUTOTUNING OF MPC

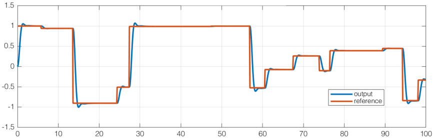
MPC CALIBRATION PROBLEM

- Controller depends on a vector x of parameters
- Parameters can be many things:
  - MPC weights, prediction model coefficients, horizons
  - Entries of covariance matrices in Kalman filter
  - Tolerances used in numerical solvers
  - ...

- Define a **performance index** f over a closed-loop simulation or real experiment. For example:

$$f(x) = \sum_{t=0}^{T} \|y(t) - r(t)\|^2$$

(tracking quality)



- **Auto-tuning** = find the best combination of parameters that solves the **global optimization problem**

$$\min_x f(x)$$

GLOBAL OPTIMIZATION ALGORITHMS FOR AUTO-TUNING

What is a good optimization algorithm to solve $\min f(x)$ **?**

- The algorithm should not require the gradient ∇f of $f(x)$ (**derivative-free** or **black-box optimization**)

- The algorithm should not get stuck on local minima (**global optimization**)

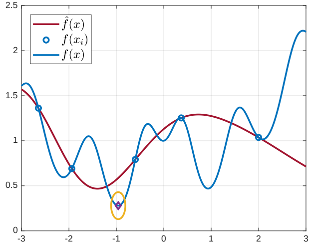
- The algorithm should make the **fewest evaluations** of the cost function f (which is expensive to evaluate)

AUTO-TUNING - GLOBAL OPTIMIZATION ALGORITHMS

- Several derivative-free global optimization algorithms exist: (Rios, Sahidinis, 2013)
  - Lipschitzian-based partitioning techniques:
    - **DIRECT** (DIvide in RECTangles) (Jones, 2001)
    - Multilevel Coordinate Search (**MCS**) (Huyer, Neumaier, 1999)
  - Response surface methods
    - **Kriging** (Matheron, 1967), **DACE** (Sacks et al., 1989)
    - Efficient global optimization (**EGO**) (Jones, Schonlau, Welch, 1998)
    - **Bayesian optimization** (Brochu, Cora, De Freitas, 2010)
  - Genetic algorithms (**GA**) (Holland, 1975)
  - Particle swarm optimization (**PSO**) (Kennedy, 2010)
  - ...
- **New method**: radial basis function surrogates + inverse distance weighting (**GLIS**) (Bemporad, 2019)

`cse.lab.imtlucca.it/~bemporad/glis`

AUTO-TUNING - GLIS



- **Goal**: solve the **global optimization** problem

$$\begin{array}{rl} \min_x & f(x) \\ \text{s.t.} & \ell \le x \le u \\ & g(x) \le 0 \end{array}$$

- **Step #0**: Get random initial samples $x_1, \ldots, x_{N_{\rm init}}$ **(Latin Hypercube Sampling)**

- **Step #1**: given N samples of f at $x_1, \ldots, x_N$, build the **surrogate function**

$$\hat{f}(x) = \sum_{i=1}^{N} \beta_i \phi(\epsilon \|x - x_i\|_2)$$

$\phi =$ radial basis function

Example: $\phi(\epsilon d) = \frac{1}{1+(\epsilon d)^2}$

(inverse quadratic)

Vector β solves $\hat{f}(x_i) = f(x_i)$ for all $i = 1, \ldots, N$ (=linear system)

- **CAVEAT**: build and minimize $\hat{f}(x_i)$ iteratively may easily miss global optimum!

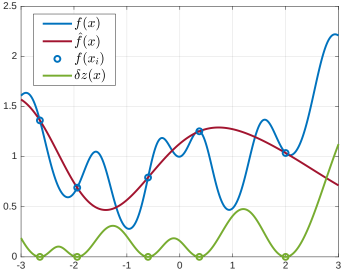
- **Step #2**: construct the **IDW exploration function**

$$z(x) = \frac{2}{\pi}\Delta F \tan^{-1}\left(\frac{1}{\sum_{i=1}^{N} w_i(x)}\right)$$

or 0 if $x \in \{x_1, \ldots, x_N\}$

where $w_i(x) = \dfrac{e^{-\|x-x_i\|^2}}{\|x-x_i\|^2}$

ΔF = observed range of $f(x_i)$



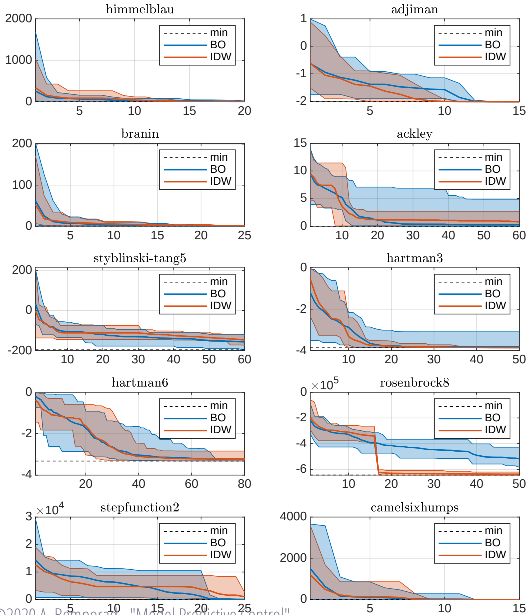
- **Step #3**: optimize the **acquisition function**

$$\begin{array}{rcl} x_{N+1} = & \arg\min & \hat{f}(x) - \delta z(x) \\ & \text{s.t.} & \ell \leq x \leq u,\ g(x) \leq 0 \end{array}$$

to get new sample x_{N+1}

δ= *exploitation* vs *exploration* tradeoff

- Iterate the procedure to get new samples $x_{N+2}, \ldots, x_{N_{\max}}$



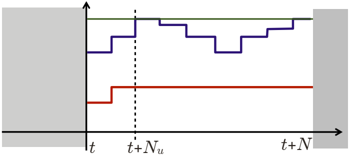
problem	n	BO[s]	IDW [s]
ackley	2	26.42	3.24
adjiman	2	3.39	0.66
branin	2	9.58	1.27
camelsixhumps	2	4.49	0.62
hartman3	3	23.19	3.58
hartman6	6	52.73	10.08
himmelblau	2	7.15	0.92
rosenbrock8	8	58.31	11.45
stepfunction2	4	10.52	1.72
styblinski-tang5	5	33.30	5.80

Results computed on 20 runs per test

BO = MATLAB's **bayesopt** fcn

- We want to auto-tune the linear MPC controller

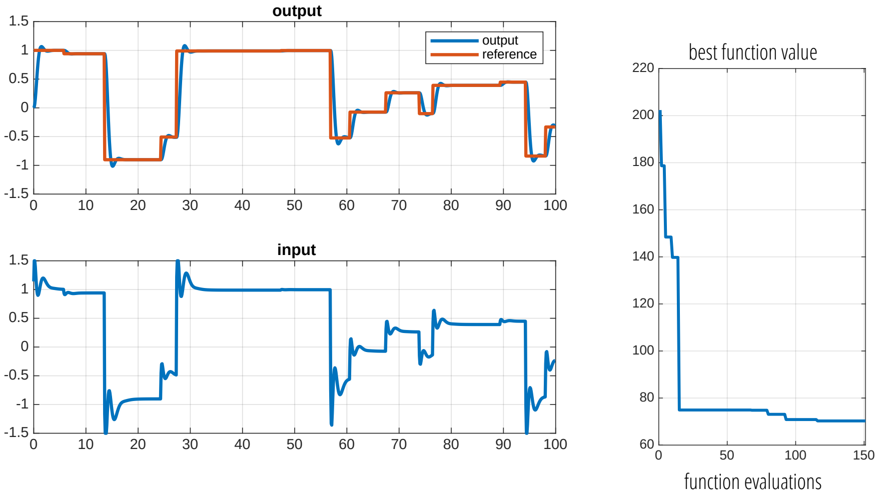
$$\begin{array}{ll}\min & \sum_{k=0}^{50-1} (y_{k+1}-r(t))^2 + ({\color{red}W^{\Delta u}}(u_k-u_{k-1}))^2 \\ \text{s.t.} & x_{k+1} = Ax_k + Bu_k \\ & y_c = Cx_k \\ & -1.5 \leq u_k \leq 1.5 \\ & u_k \equiv u_{N_u},\ \forall k = {\color{red}N_u}, \ldots, N-1\end{array}$$



- Calibration parameters: $x = [\log_{10} W^{\Delta u}, N_u]$
- Range: $-5 \leq x_1 \leq 3$ and $1 \leq x_2 \leq 50$
- Closed-loop performance objective:

$$f(x) = \sum_{t=0}^{T} \underbrace{(y(t)-r(t))^2}_{\text{track well}} + \underbrace{\frac{1}{2}(u(t)-u(t-1))^2}_{\text{smooth control action}} + \underbrace{2N_u}_{\text{small QP}}$$

AUTO-TUNING: EXAMPLE



- Result: $x^\star = [-0.2341, 2.3007]$ $\Rightarrow$ $W^{\Delta u} = 0.5833, N_u = 2$

AUTO-TUNING: PROS AND CONS

- Pros:
  - Selection of calibration parameters x to test is fully automatic
  - Applicable to any calibration parameter (weights, horizons, solver tolerances, ...)

    (Piga, Forgione, Formentin, Bemporad, 2019) (Forgione, Piga, Bemporad, 2020)
  - Rather arbitrary performance index $f(x)$ (tracking performance, response time, worst-case number of flops, ...)
- Cons:
  - Need to **quantify** an objective function $f(x)$
  - No room for **qualitative** assessments of closed-loop performance
  - Often objectives are multiple, not clear how to blend them in a **single** one
- Current research: **preference-based optimization** (**GLISp**), having human assessments in the loop (**semi-automatic tuning**)

  (Bemporad, Piga, 2019)

  (Zhu, Bemporad, Piga, 2020)

`cse.lab.imtlucca.it/~bemporad/glis`

LEARNING MPC FROM DATA - LESSON LEARNED SO FAR

- 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