LEARNING-BASED METHODS FOR NONLINEAR AND GAME-THEORETIC MODEL PREDICTIVE CONTROL

Alberto Bemporad

imt.lu/ab

Funded by the European Union

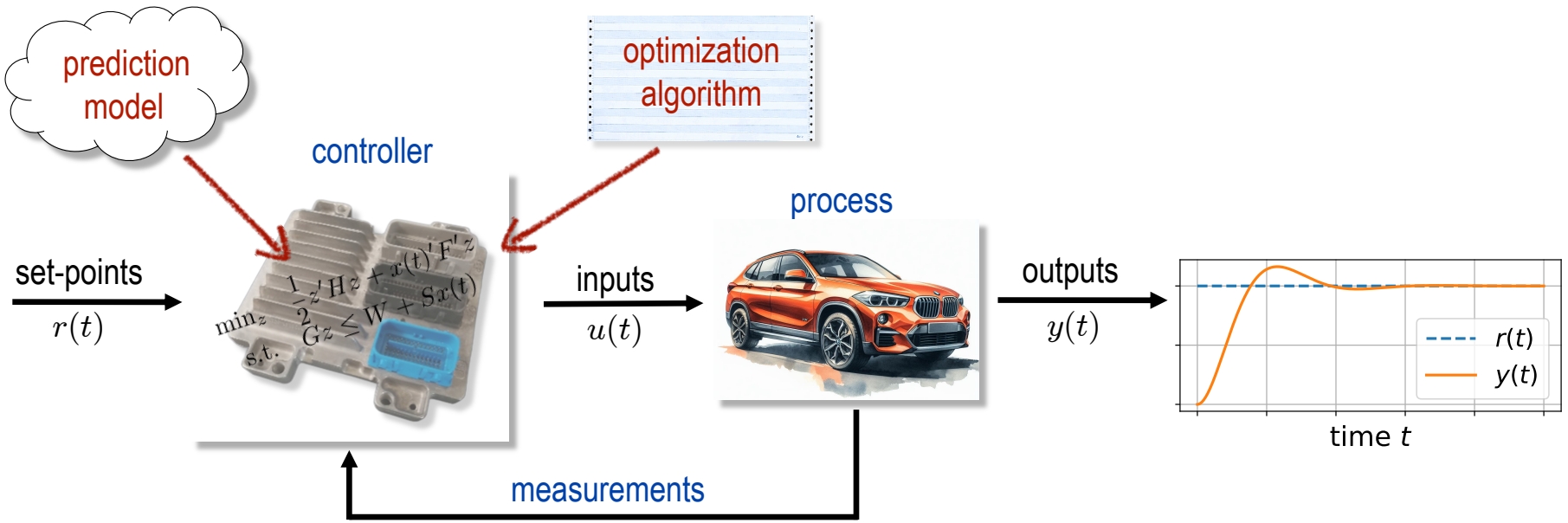
European Research Council
Established by the European Commission

ERC Advanced Grant "COMPACT"

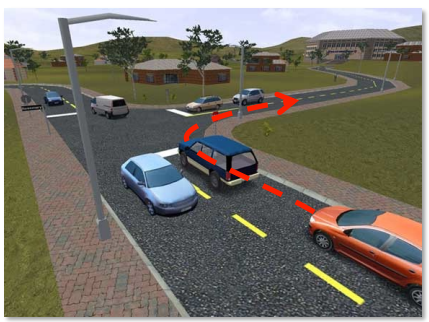
IMT SCHOOL FOR ADVANCED STUDIES LUCCA

TU Darmstadt, October 27, 2026

MODEL PREDICTIVE CONTROL (MPC)



- Use a dynamical **Model** of the process to **Predict** its future evolution and choose the "best" **Control** action accordingly



MODEL PREDICTIVE CONTROL

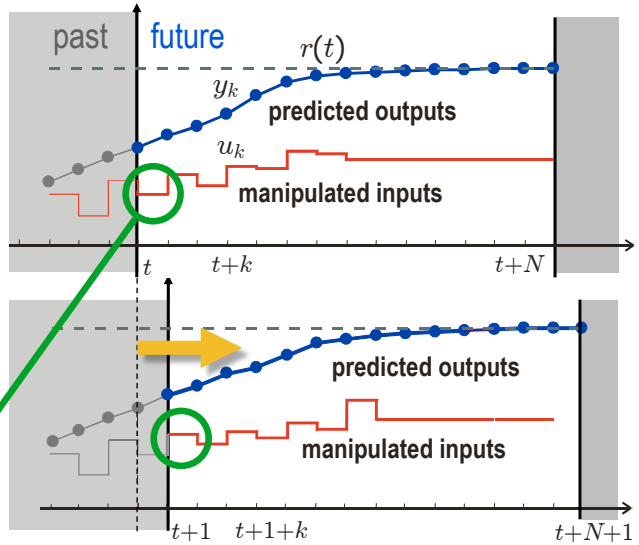
- MPC algorithm:

$$\min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} \|y_k - r(t)\|_2^2 + \rho\|u_k - u_r(t)\|_2^2$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k) \qquad \text{prediction model}$$
$$y_k = g(x_k)$$

$$u_{\min} \le u_k \le u_{\max} \qquad \text{constraints}$$
$$y_{\min} \le y_k \le y_{\max}$$

$$x_0 = x(t) \qquad \text{state feedback}$$



numerical optimization problem

1. **estimate** current state $x(t)$
2. **optimize** wrt $\{u_0, \dots, u_{N-1}\}$
3. **apply** only the optimal u_0 as input $u(t)$

Repeat at each time step t

MPC IN INDUSTRY

- Conceived in the 1960s (Rafal, Stevens, 1968) (Propoi, 1963)

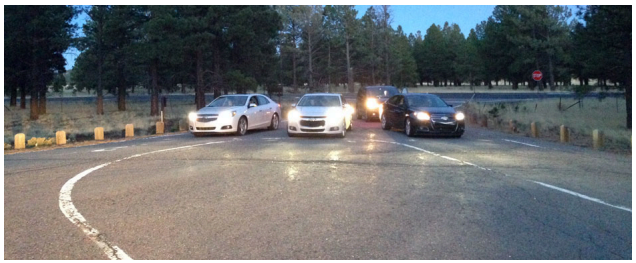




- Used in the **process industries** since the 1980s (Qin, Badgwell, 2003)
- Now massively spreading to the automotive industry and other sectors
- MPC by **General Motors** and **ODYS** in high-volume production since 2018 (**3+ million vehicles worldwide**)

(Bemporad, Bernardini, Long, Verdejo, 2018)

(Bemporad, Bernardini, Livshiz, Pattipati, 2018)



First known mass production of MPC in the automotive industry

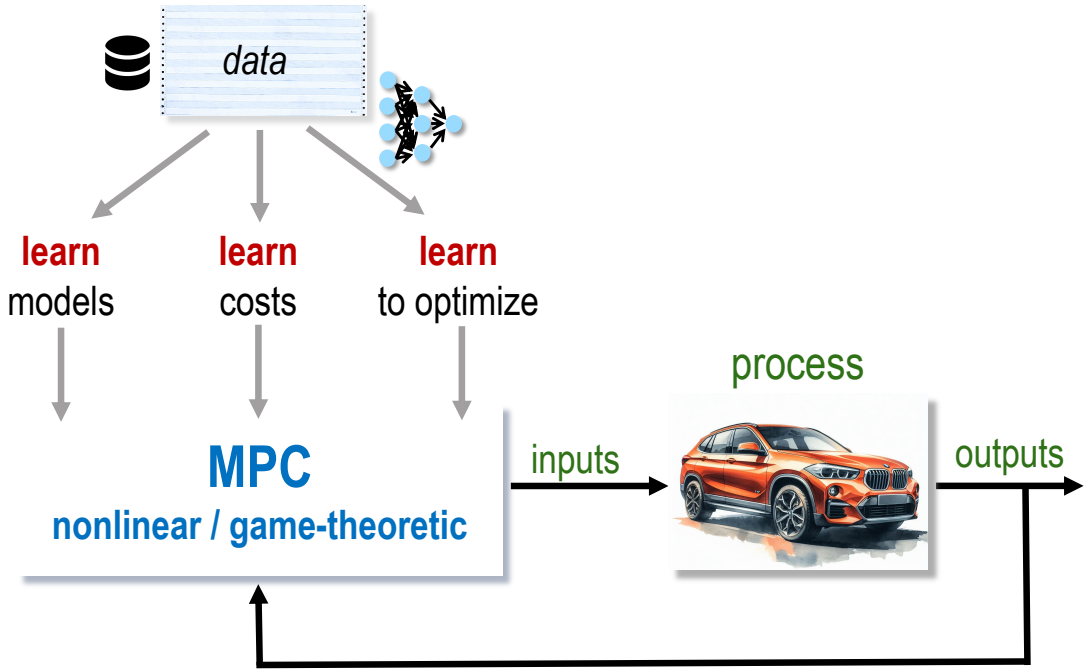
and many more are underway ...

ODYS
Advanced Controls & Optimization

`http://www.odys.it/odys-and-gm-bring-online-mpc-to-production`

FOCUS OF THIS TALK

How can we leverage **learning methods** to help with MPC design?

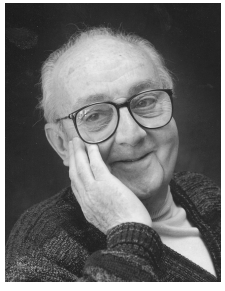


1. Learn nonlinear **dynamical models** for prediction
2. Learn MPC **cost functions** to minimize
3. Learn to **optimize** the control actions

LEARNING NONLINEAR PREDICTION MODELS

"All models are wrong, but some are useful."

(George E. P. Box)



CONTROL-ORIENTED MODELS

- **Complex model $\Rightarrow$ complex MPC controller**!
- Typically, use **small-scale** models (e.g., ≤ 10 states/inputs/outputs) with a limited number of parameters (10^3 vs. 10^{12} parameters in LLMs)
- **Limit nonlinearities** as much as possible (avoid very deep neural networks)
- Need to find the **best model** within a **poor model class**
  (= reduced risk of overfitting, given sufficient excitation)

Learning control-oriented models requires different algorithms than typical machine learning tasks

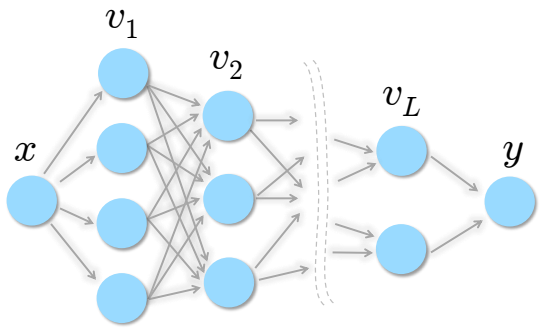
RECURRENT NEURAL NETWORKS

- **Neural networks** proposed for nonlinear system identification since the 1990s

  (Narendra, Parthasarathy, 1990) (Hunt et al., 1992) (Suykens, Vandewalle, De Moor, 1996)

- **Recurrent Neural Network** (RNN) model:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + f_x(x_k, u_k, \theta_x) \\ \hat{y}_k &= Cx_k + Du_k + f_y(x_k, \theta_y) \\ f_x, f_y &= \text{feedforward neural networks} \end{aligned}$$



- **Training problem**: given an I/O dataset $\{u_0, y_0, \ldots, u_{N-1}, y_{N-1}\}$ solve

$$\min_{\substack{\theta = (A, B, C, D, \theta_x, \theta_y) \\ x_0, x_1, \ldots, x_{N-1}}} \quad r(x_0, \theta) + \frac{1}{N} \sum_{k=0}^{N-1} \ell(y_k, \hat{y}_k)$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k + f_x(x_k, u_k, \theta_x)$$

- **Main issue**: x_k are **hidden states** and hence also **unknowns** of the problem

OPTIMIZATION METHODS FOR TRAINING RNNS

- Problem **condensing**: substitute $x_{k+1} = f_x(x_k, u_k, \theta_x)$ recursively and solve

$$\min_{\theta, x_0} \quad r(x_0, \theta) + \frac{1}{N} \sum_{k=0}^{N-1} \ell(y_k, \hat{y}_k) \quad = \quad \min_{\theta, x_0} V(\theta, x_0)$$

- **Gradient descent** (**GD**) methods: update θ, x_0 by setting

$$\begin{bmatrix} \theta^{t+1} \\ x_0^{t+1} \end{bmatrix} = \begin{bmatrix} \theta^t \\ x_0^t \end{bmatrix} - \alpha_t \nabla V(\theta^t, x_0^t)$$

Example: **Adam** uses adaptive moment estimation to set the learning rate α_t

(Kingma, Ba, 2015)

  - **Pros**: no line search $\rightarrow$ can help escape local minima
  - **Cons**: **slow convergence** rate (both in theory and in practice)

OPTIMIZATION METHODS FOR TRAINING RNNS

- **Stochastic** gradient descent (SGD) is not well suited for training RNNs:
  - collect a large number of short independent experiments (often impossible)
  - or create mini-batches by using **multiple-shooting** ideas

    (Forgione, Piga, 2020) (Bemporad, 2023)

- **Newton's method**: very fast (2^{nd}-order) local convergence but difficult to implement, as we need **Hessian** $\nabla^2 V(\theta^t, x_0^t)$ information

- **Quasi-Newton methods**: good convergence speed / solution tradeoff, only the **gradient** $\nabla V(\theta^t, x_0^t)$ required (e.g., **L-BFGS**[1]) (Liu, Nocedal, 1989)

- **Extended Kalman Filtering** (EKF): a recursive Gauss-Newton method for learning nonlinear models **online**

[1] Limited-memory Broyden–Fletcher–Goldfarb–Shanno

A PYTHON LIBRARY FOR SYSTEM IDENTIFICATION

(Bemporad, 2025)

- Python package to identify **linear/nonlinear/static** models:

jax-sysid

```
from jax_sysid.models import Model

def state_fcn(x,u,params):
    ...

def output_fcn(x,u,params):
    ...

model = Model(nx, ny, nu, state_fcn=state_fcn, output_fcn=output_fcn)

model.init(params=[A,B,C,W1,W2,W3,b1,b2,W4,W5,b3,b4])
model.loss(rho_x0=1.e-4, rho_th=1.e-4, tau_th=1.e-4)
model.optimization(adam_epochs=1000, lbfgs_epochs=1000)
model.fit(Y, U)

Yhat, Xhat = model.predict(model.x0, U)
```

state-update function, $x(k+1)$

output function, $y(k)$



```
pip install jax-sysid
```

```
github.com/bemporad/jax-sysid
```

- Models trained by cascaded **Adam + L-BFGS-B** optimization

QUASI-LPV PREDICTION MODELS

(Shamma, Athans, 1991) (Tóth, 2010)

- **Quasi-Linear Parameter-Varying** (qLPV) models:

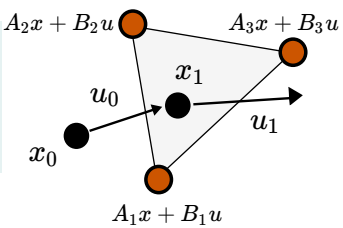
$$\begin{aligned} x_{k+1} &= A(p(x_k))x_k + B(p(x_k))u_k \\ \hat{y}_k &= Cx_k \\ p(x_k) &= \text{softmax}(\mathcal{N}_1(x_k), \ldots, \mathcal{N}_{n_p}(x_k)) \end{aligned}$$

$$A(p) = \sum_{i=1}^{n_p} A_i p_i,\ B(p) = \sum_{i=1}^{n_p} B_i p_i$$

$$\Rightarrow \quad 0 \le p_{ki} \le 1, \quad \sum_{i=1}^{n_p} p_{ki} = 1$$

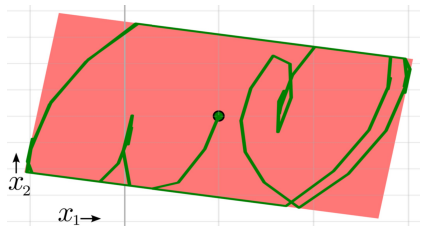
```
from jax_sysid.models import qLPVModel

model = qLPVModel(nx, ny, nu, npar, qlpv_fcn, qlpv_params_init)
```



- Useful to **recast NL-MPC** into **linear MPC** by fixing p_k in prediction, or **robust linear MPC** by modeling $x_{k+1} \in \text{conv}(A_i x_k + B_i u_k)$

- Joint learning of a **qLPV model** + **robust control invariant set** is possible (Mulagaleti, Bemporad, 2025)



MODEL ADAPTATION AND OFFSET-FREE TRACKING

ONLINE MODEL ADAPTATION

(Puskorius, Feldkamp, 1994) (Wang, Huang, 2011)

- **Key idea**: treat the parameters of the feedforward NNs as **constant states** and iterate an **extended Kalman filter** (EKF) on the training dataset:

$$\begin{cases} x_{k+1} &= f_x(x_k, u_k, \theta_{xk}) + \xi_k \\ \begin{bmatrix} \theta_{x(k+1)} \\ \theta_{y(k+1)} \end{bmatrix} &= \begin{bmatrix} \theta_{xk} \\ \theta_{yk} \end{bmatrix} + \eta_k \\ \hat{y}_k &= f_y(x_k, \theta_{yk}) + \zeta_k \end{cases}$$

$$Q = \mathrm{Var}[\eta_k]$$
$$R = \mathrm{Var}[\zeta_k]$$
$$P_0 = \mathrm{Var}\begin{bmatrix} \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} \\ x_0 \end{bmatrix}$$

- EKF provides both an estimate of θ_x, θ_y (and its associated uncertainty via the **covariance matrix** P_k), and a consistent state estimate x_k

- The ratio Q/R determines the **learning rate** of the training algorithm

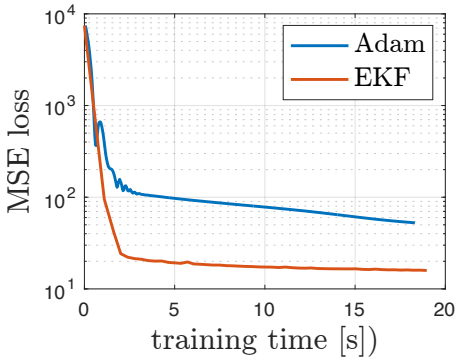
- Also applies to gray-box models (physical parameters + neural nets)

ONLINE MODEL ADAPTATION

- EKF extended to handle **general strongly convex** and smooth losses $\ell(y_k, \hat{y}_k)$ and **nonsmooth regularization** (e.g., ℓ_1**-sparsifiers**) (Bemporad, 2023)

- Model can be estimated **online** from streaming data $\{u_k, y_k\}$

- EKF can also be used for **offline** system identification



- **Alternative #1**: train via **Moving Horizon Estimation** (MHE) instead of EKF (Løwenstein, Bernardini, Bemporad, Fagiano, 2023)

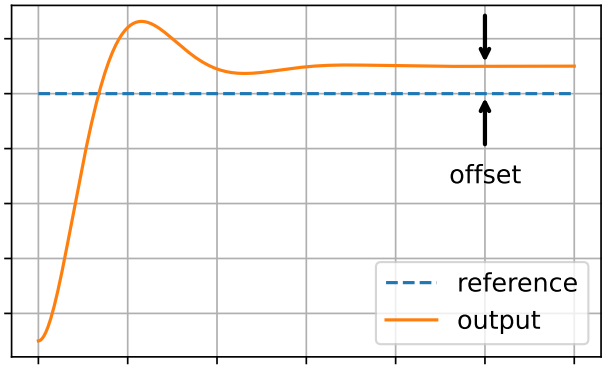
- **Alternative #2**: use EKF + ADMM to handle more general nonsmooth penalties (e.g., ℓ_0**-sparsifiers**) and constraints (Frascati, Bemporad, 2026)

DISTURBANCE MODELS FOR OFFSET-FREE TRACKING

- **Tracking errors** in steady state can be due to **model mismatch** / **disturbances**

$$\begin{cases} x_{k+1} &= Ax_k + Bu_k \\ \hat{y}_k &= Cx_k \end{cases}$$

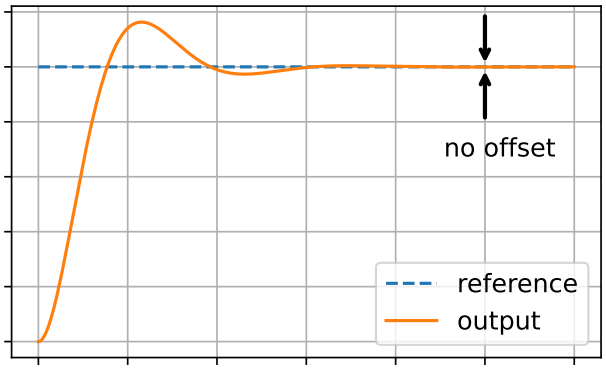
prediction model $\neq$ plant dynamics



- **Possible remedy**: introduce a **constant disturbance model**

$$\begin{cases} x_{k+1} &= Ax_k + Bu_k \\ d_{k+1} &= d_k \\ \hat{y}_k &= Cx_k + d_k \end{cases}$$

augmented prediction model



and estimate x_k, d_k by a **state observer**

(Pannocchia, Gabiccini, Artoni, 2015)

ONLINE LEARNING OF DISTURBANCE MODELS

- Training of the full model online can be hard (lack of excitation, CPU time, safety, …)

- **Key idea**: only train a **disturbance model** online to refine the prediction model only where the system is operating

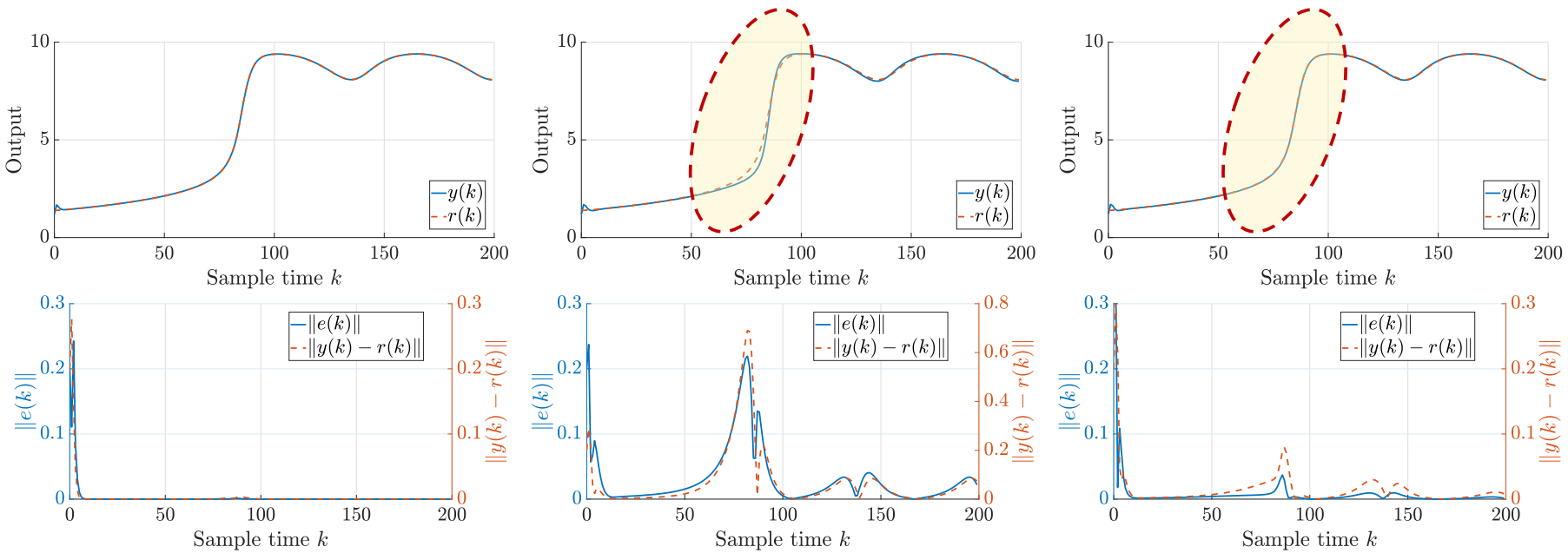
$$\begin{aligned} x(k+1) &= f(x(k), u(k), d(k)) \\ d(k) &= h(x(k), u(k), \theta(k)) \\ y(k) &= g(x(k), d(k)) \end{aligned}$$

- The nominal model f, g is trained offline and frozen

- Under reasonable assumptions, the tracking error $y(k) - r(k)$ **converges to zero**, **even if $r(k)$ is not constant** (Krupa, Zanon, Bemporad, 2025)

EXAMPLE: CHEMICAL REACTOR

- NL-MPC of a **continuous stirred-tank reactor (CSTR)** (Bequette, 1998)
- Reference signal $r(k)$ (**w/ preview**), trackable
- **Standard approach**: **constant** disturbance models (Pannocchia, Gabiccini, Artoni, 2015)



(a) White-box disturbance model (b) Constant disturbance model (c) FNN disturbance model

- Better tracking with **FNN** disturbance model than constant disturbance model

LEARNING MPC COST FUNCTIONS AND CALIBRATION

MPC CALIBRATION PROBLEM

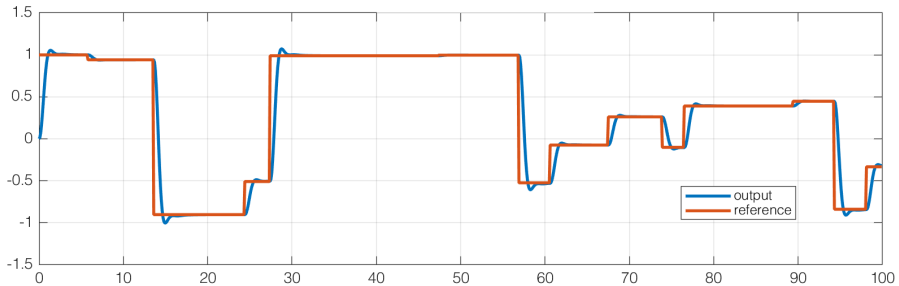
- The design depends on a vector x of **MPC parameters**



- Parameters can be many things:
  - MPC weights, prediction model coefficients, horizons
  - Covariance matrices used in Kalman filters
  - 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)



- **Automatic calibration** = find the **best** combination of parameters by solving the **global optimization problem**

$$\min_x f(x)$$

ACTIVE PREFERENCE LEARNING

- The objective function $f(x)$ may **not be** available (multiple objectives, qualitative assessments/human judgement, ...)
- We can only express a **preference** between two choices:

$$\pi(x_1, x_2) = \begin{cases} -1 & \text{if } x_1 \text{ "better" than } x_2 \quad [f(x_1) < f(x_2)] \\ 0 & \text{if } x_1 \text{ "as good as" } x_2 \quad [f(x_1) = f(x_2)] \\ 1 & \text{if } x_2 \text{ "better" than } x_1 \quad [f(x_1) > f(x_2)] \end{cases}$$

- We want to find a global optimum $x^\star$ (="better" than any other x)

$$\text{find } x^\star \text{ such that } \pi(x^\star, x) \leq 0,\ \forall x \in \mathcal{X},\ \ell \leq x \leq u$$

- **Active preference learning**: iteratively propose a new sample to compare
- **Key idea**: learn a **surrogate** of the (latent) objective function from preferences

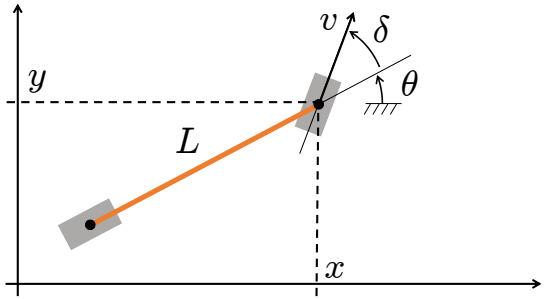
(Bemporad, Piga, 2021)

PREFERENCE-BASED TUNING: MPC EXAMPLE

(Zhu, Bemporad, Piga, 2021)

- Example: calibration of a simple MPC for lane-keeping (2 inputs, 3 outputs)

$$\begin{cases} \dot{x} &= v\cos(\theta+\delta) \\ \dot{y} &= v\sin(\theta+\delta) \\ \dot{\theta} &= \frac{1}{L}v\sin(\delta) \end{cases}$$



- Multiple control objectives:

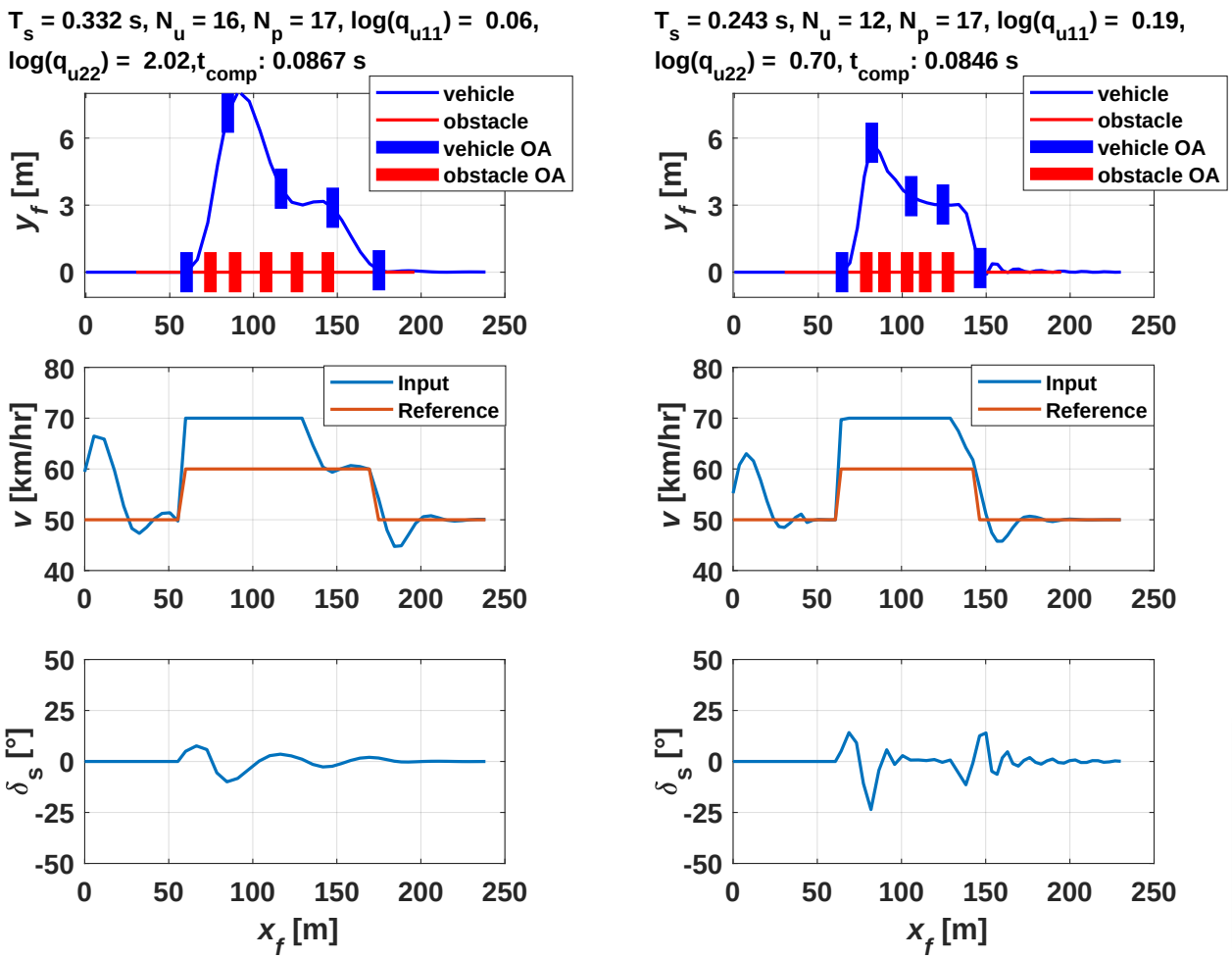
  "*optimal obstacle avoidance*", "*pleasant drive*", "*CPU time small enough*", …

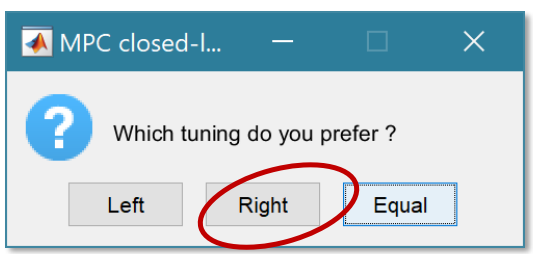
  ⇒ **not easy to quantify in a single function**

- 5 MPC parameters to tune:
  - **sampling time**
  - prediction and control **horizons**
  - **weights** on input increments Δv, $\Delta\delta$

PREFERENCE-BASED TUNING: MPC EXAMPLE

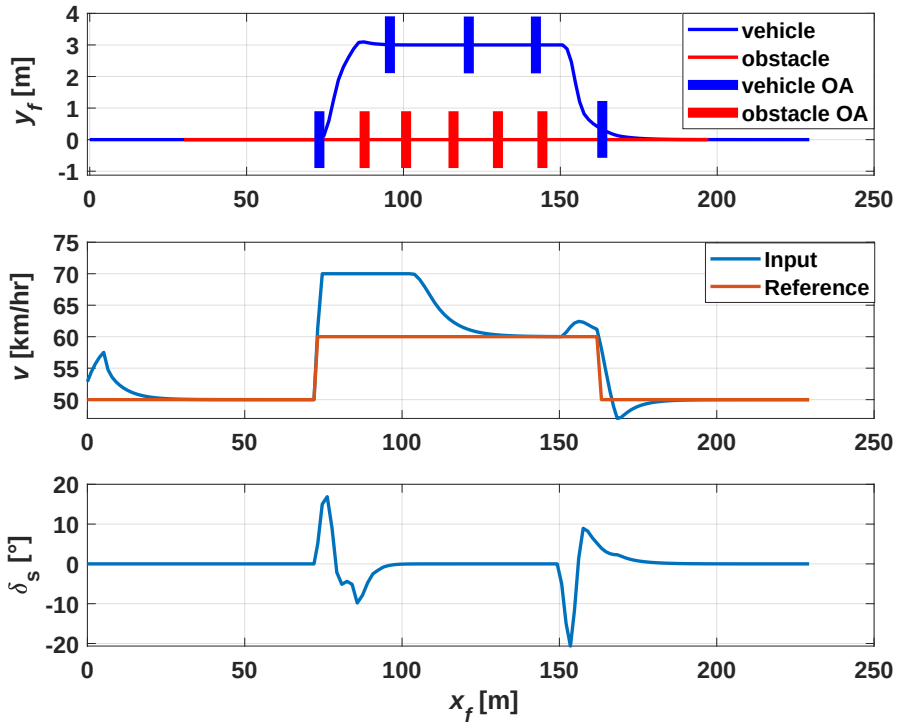
- Preference query window:





PREFERENCE-BASED TUNING: MPC EXAMPLE

- Convergence after 50 GLISp iterations (=49 queries):



Optimal MPC parameters:

- sample time = 85 ms (CPU time = 80.8 ms)
- prediction horizon = 16
- control horizon = 5
- weight on Δv = 1.82
- weight on $\Delta\delta$ = 8.28



- **Note**: no need to define a closed-loop performance index explicitly!
- Python package:

```
pip install glis
```

PREFERENCE-BASED LEARNING OF MPC COSTS

(Krupa, El Hasnaouy, Zanon, Bemporad, 2025)

- **Alternative**: learn the **MPC cost** directly by **comparing trajectories**

  X = state trajectory, U = input trajectory. Let $T = (X, U)$

- Comparisons are evaluated (manually or automatically) via a **preference function**

$$\pi(T_i, T_j) = \begin{cases} 1 & \text{if } T_i \text{ is preferred to } T_j \\ 0 & \text{otherwise} \end{cases}$$

- **Key idea**: there exists an (unmeasurable) **latent function** $\sigma(T)$ such that

$$\pi(T_i, T_j) = \begin{cases} 1 & \text{if } \sigma(T_i) \leq \sigma(T_j) \\ 0 & \text{otherwise} \end{cases}$$

- **Procedure**: collect preferences $(T_i, T_j, \pi(T_i, T_j))$ and fit a **binary classifier**

$$\hat{\pi}(T_i, T_j, \theta) = \frac{1}{1 + \exp\left(\hat{\sigma}(T_i, \theta) - \hat{\sigma}(T_j, \theta)\right)} \qquad \hat{\sigma}(T, \theta) = \textbf{convex} \text{ wrt } T$$

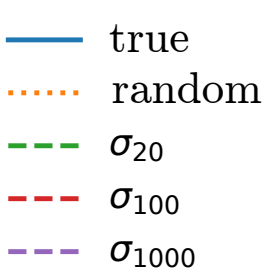
PREFERENCE-BASED LEARNING OF MPC COSTS

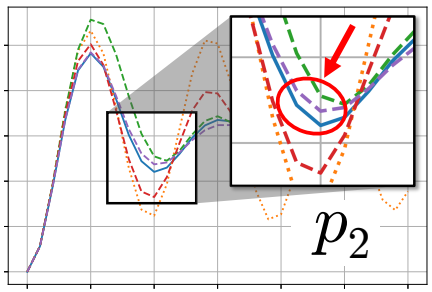
- **Learned MPC controller**: at each time t, given $x(t)$, solve the **convex problem**

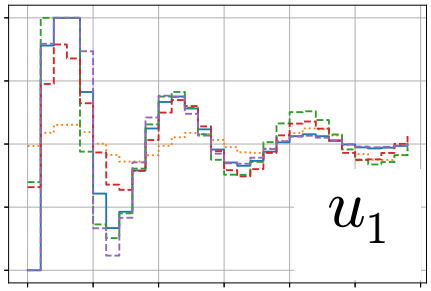
$$
\begin{aligned}
T_t \in \arg\min_T \; & \sigma(T, \theta^*) \\
\text{s.t. } & x_{k+1} = Ax_k + Bu_k, \quad k = 0, \dots, N-1 \\
& x_0 = x(t), \quad u_k \in \mathcal{U}, \quad x_k \in \mathcal{X} \\
& T = (X, U), \quad X = [x_0, \dots, x_N], \quad U = [u_0, \dots, u_{N-1}]
\end{aligned}
$$

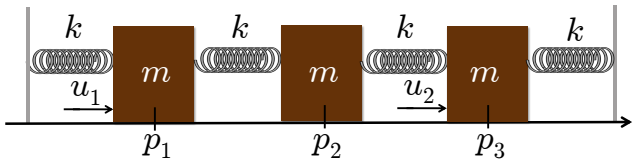
and apply $u(t)$ = first control move in T_t [$\mathcal{U}, \mathcal{X}$ are convex sets]

- **Example**: **3 oscillating masses**. Latent function $\sigma(T) = \sum_{k=0}^{N-1} x_k' Q x_k + u_k' R u_k$









Training datasets: 20, 100, 1000 comparisons generated from 50 trajectories

LEARNING TO OPTIMIZE

HOW TO SIMPLIFY MPC OPTIMIZATION?

Let $x = (u_0, \ldots, u_{N-1})$, p = current state/ref, $f(x, p)$ = MPC cost

1. Approximate the **optimizer**
   - **explicit**: learn only the first component $\hat{u}_0(p) \approx u_0^\star(p)$, and **apply it**
   - **implicit**: learn $\hat{x}(p) \approx x^\star(p)$ and use it to **warm start** an online solver

2. Approximate the **optimization problem**:
   - **black-box**: learn a simpler $\hat{f}(x, p) \approx f(x, p)$, minimize $\hat{f}(x, p)$ online wrt x
   - **gray-box**: learn a simpler MPC formulation
     - MPC with horizon $N = 1$ + learned **cost-to-go** (Abdufattokhov, Zanon, Bemporad, 2024)
     - Learn a **QP approximation** QP(p) of the NL-MPC problem (Pillitteri, Bemporad, CDC 2026)

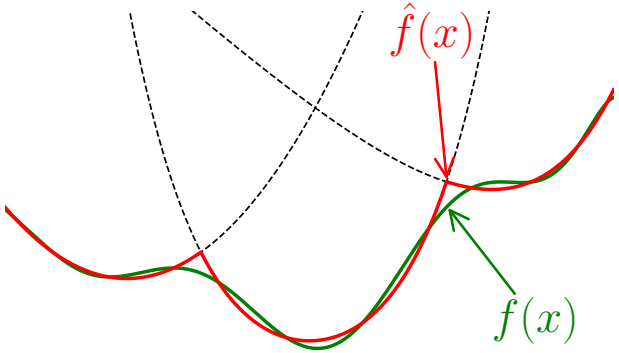
MIN-OF-CONVEX SURROGATE

(Wang, Patrinos, Bemporad, CDC 2026)

- **Main idea**: parameterize the surrogate $\hat{f}$ as the **minimum** of **convex** functions

$$\hat{f}(x,p) = \min_{i=1,\dots,K} \hat{f}_i(x,p)$$

each $\hat{f}_i(\cdot,p)$ **convex** in x



- $\hat{f}(x,p)$ captures the **nonconvexity** of $f(x,p)$ and is **easy to minimize**

- **More generally**: $\hat{f}_i$ = a **monotonically increasing** function of a **convex** function:

$$\hat{f}(x,p) = \min_{i=1,\dots,K} h_i\big(\bar{f}_i(x,p)\big) \qquad h_i : \mathbb{R} \to \mathbb{R} \quad \text{monotonically increasing}$$

LEARNING PROBLEM AND REGULARIZATION

- **Dataset**: $\{x_k, p_k, f_k\}_{k=1}^N, \quad f_k = f(x_k, p_k)$ (explicit form of f not required)

- **Unknowns**: Θ = parameters of all functions $\bar{f}_i$ and h_i to learn

- **Loss**:

$$\underbrace{\rho_1 \mathcal{R}_1(\Theta)}_{\ell_2 \text{ or } \ell_1 \text{ regularization}} + \frac{1}{N}\sum_{k=1}^{N} \left(f_k - \hat{f}_\Theta(x_k, p_k)\right)^2$$

- **Optionally**, if **gradients** $\nabla_x f_k$ also available, add **gradient-matching** loss

$$\mathcal{R}_2(\Theta) = \frac{\rho_2}{N}\sum_{k=1}^{N} \left\|\nabla_x \hat{f}_\Theta(x_k, p_k) - \nabla_x f_k\right\|_2^2$$

EXAMPLE: NONCONVEX FUNCTION APPROXIMATION

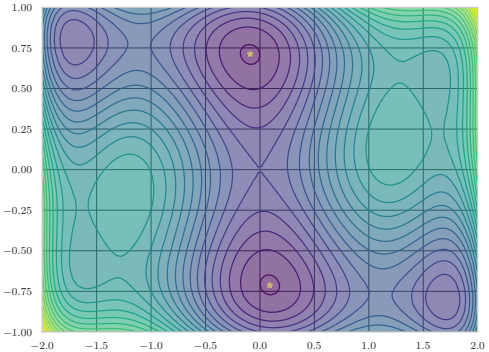
- **Six-hump camel-back** function

$$\bar{f}_i = \sum_{j=1}^{L} \max\{A_j x + b_j, 0\}^2$$

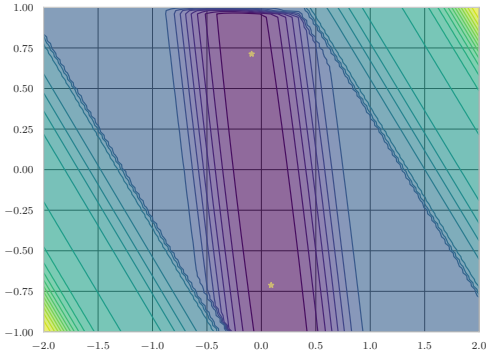
h_i = small monotonic NN (shared)

$N = 1000$ samples

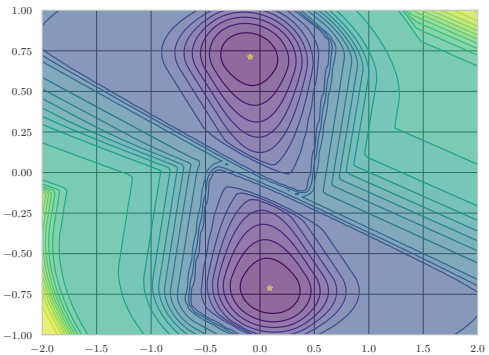
training time = 7.4 s [Apple M5 Max]



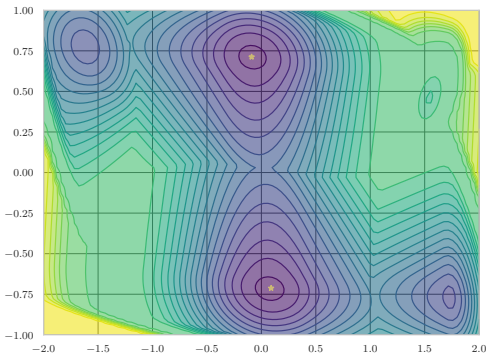
ground truth



$K = 1$



$K = 2$



$K = 5$

- **Note**: minimizing the **surrogate** ($K = 5$) directly lands near the global min, while only 64 out of 100 random L–BFGS starts on the **original** function do

GAME-THEORETIC MPC

GENERALIZED NASH EQUILIBRIUM PROBLEM

- N interacting (**non-cooperative**) agents, each with its own objective J_i
- Agent #i decides $x_i \in \mathbb{R}^{n_i}$ given the other agents' decisions $x_{-i} \in \mathbb{R}^{n-n_i}$:

$$\bar{x}_i(x_{-i}, p) = \underset{x_i:\ (x_i, x_{-i}) \in \mathcal{X}(p)}{\arg\min} J_i(x_i, x_{-i}, p)$$

$\bar{x}_i$ = **best response** (assumed unique)

- (Parametric) **Generalized Nash Equilibrium** (GNE):

$$x_i^\star(p) = \underset{x_i:\ (x_i, x^\star_{-i}(p)) \in \mathcal{X}(p)}{\arg\min} J_i(x_i, x^\star_{-i}(p), p) \quad \forall i = 1, \dots, N$$

- A unique variational GNE $x^\star(p)$ exists under suitable assumptions on J_i's and $\mathcal{X}$. In particular, **strong monotonicity** of the **pseudo-gradient** (Facchinei, Pang, 2003)

$$F(x,p) \triangleq \begin{bmatrix} \nabla_{x_1} J_1(x_1, x_{-1}, p) \\ \vdots \\ \nabla_{x_N} J_N(x_N, x_{-N}, p) \end{bmatrix}$$

$$\frac{1}{2}(\nabla_x F(x,p) + \nabla_x F(x,p)') \succeq \mu I \quad \mu > 0$$

$\nabla_x F(x,p)$ = pseudo-gradient's Jacobian

GAME-THEORETIC MPC

- **Game-theoretic MPC** problem with N agents and input/output constraints:

$$\begin{aligned}
\min_{\Delta u_i,\epsilon_i} \quad & \left(\sum_{k=0}^{T-1}(y_{k+1}-r(t))^\top Q_{y,i}(y_{k+1}-r(t))+\Delta u_{k,i}^\top Q_{\Delta u,i}\Delta u_{k,i}\right)+q_{\epsilon,i}\epsilon_i \\
\text{s.t.} \quad & x_{k+1}=Ax_k+Bu_k \qquad\qquad y_{k+1}=Cx_{k+1} \\
& u_{k,i}=u_{k-1,i}+\Delta u_{k,i} \qquad\qquad u_{-1}=u(t-1) \\
& \Delta u_{\min}\le \Delta u_k \le \Delta u_{\max} \qquad\qquad u_{\min}\le u_k\le u_{\max} \\
& y_{\min}-\textstyle\sum_{i=1}^N\epsilon_i \le y_{k+1}\le y_{\max}+\sum_{i=1}^N\epsilon_i \qquad\qquad \epsilon_i\ge 0 \\
& i=1,\dots,N,\ k=0,\dots,T-1.
\end{aligned}$$

- **Condense** the problem by substituting $x_k = A^k x_0 + \sum_{j=0}^{k-1} A^{k-1-j}Bu_j$
- GT-MPC problem = **Linear-Quadratic GNE** problem
- Problem solved at each time t via the `nashopt` Python package (Bemporad, 2026)



```
pip install nashopt
```

NashOpt

EXAMPLE: GAME-THEORETIC MPC

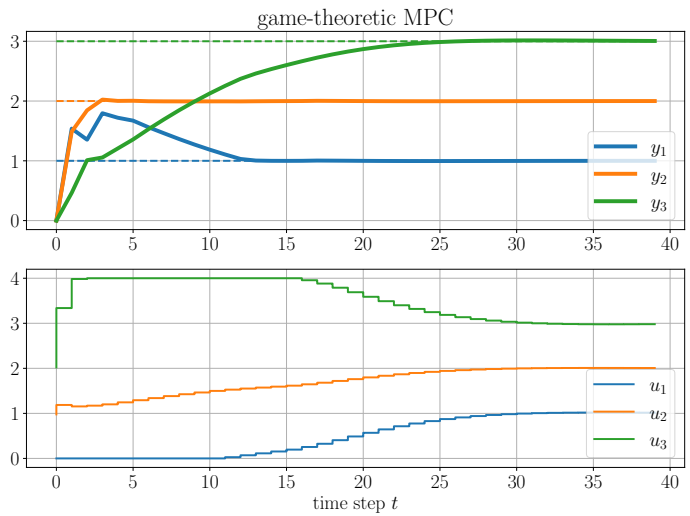
- Linear system: $x \in \mathbb{R}^6, u \in \mathbb{R}^3, y \in \mathbb{R}^3$. Agent i controls u_i and weights only y_i

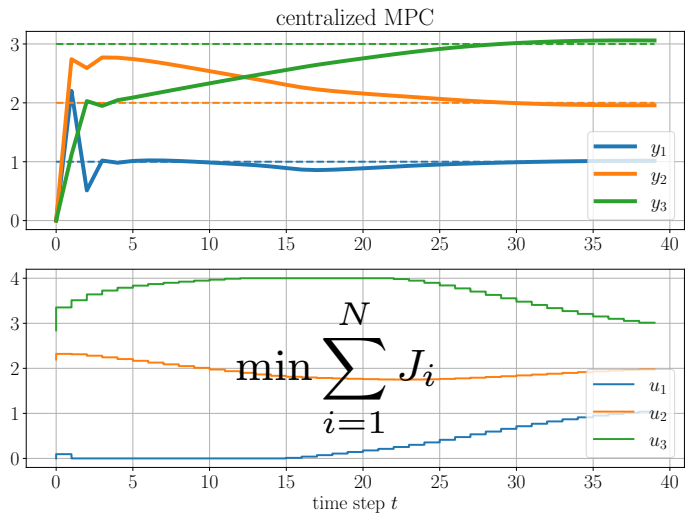
```
from nashopt import NashLinearMPC

nash_mpc = NashLinearMPC(sizes, A, B, C, Qy, Qdu, T=10, ymin=ymin, ymax=ymax,
            umin=umin, umax=umax, dumin=dumin, dumax=dumax, Qeps=Qeps, Tc=3)
```

- At each time t, compute a variational GNE via MILP and apply $u(t)$

```
sol = nash_mpc.solve(x, u1, r)
u = sol.u
```





CPU time: **4.36-65.62 ms** (GNE via HiGHS MILP) vs **0.57-2.80 ms** (centralized MPC via OSQP)

STABILIZING GAME-THEORETIC MPC

(Dimou, Fabiani, Bemporad, CDC 2026)

- Linear system, linear **shared constraint** $x_k \in \mathcal{X}$ and local constraints $u_{k,i} \in \mathcal{U}_i$
- Agent #i minimizes $J_i(\mathbf{u}, x_0)$, convex wrt its own input sequence $\mathbf{u}_i$
- **maximal λ-contractive control invariant set** $\mathcal{X}_\infty^\lambda = \{x : \; Fx \leq \mathbb{1}\}$

  $\Rightarrow$ **control Lyapunov function** $V_\lambda(x) = (\max_j F_j x)^2$

$$\forall x \in \mathcal{X}_\infty^\lambda \quad \exists u \in \mathcal{U} : \quad V_\lambda(Ax + Bu) \leq \lambda^2 V_\lambda(x) \quad \lambda \in (0,1)$$

- Imposing $V_\lambda(Ax_0 + Bu_0) \leq \lambda^2 V_\lambda(x_0) \Rightarrow$ **shared linear** constraint on u_0:

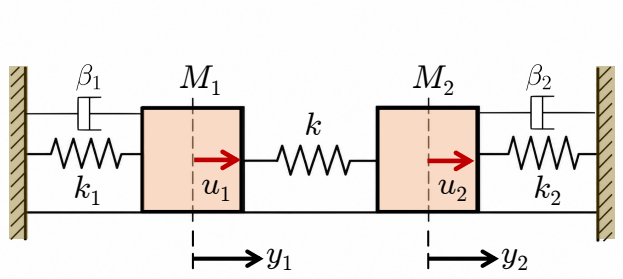
$$F(Ax_0 + Bu_0) \leq \lambda \, \mathbb{1} \max_j (F_j x_0)$$

- **Result**: if $x(0) \in \mathcal{X}_\infty^\lambda$, a GNE **exists at all times** t and $\|x(t)\| \leq \lambda^t \sqrt{c_2/c_1} \|x(0)\|$

  $\Rightarrow$ **exponential stability**, regardless of the costs J_i and of the GNE selected

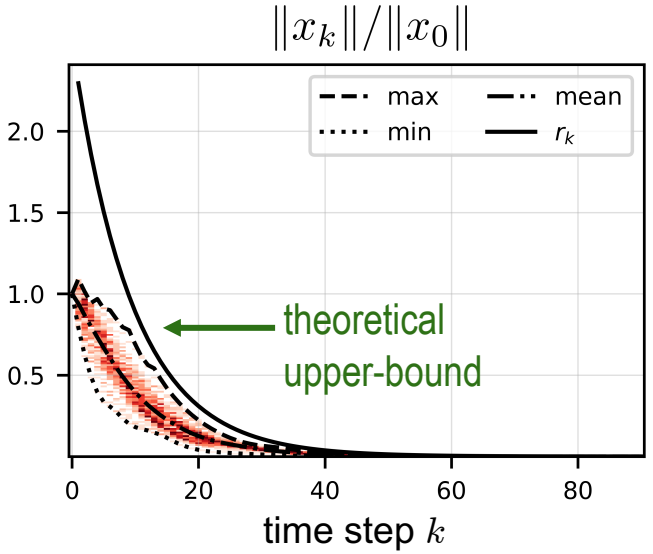
EXAMPLE: STABILIZING GAME-THEORETIC MPC

- Spring-mass system, agent #i applies force u_i to bring y_i to zero, $i = 1, 2$



- $x = [y_1\ \dot{y}_1\ y_2\ \dot{y}_2]'$, sampling time $T_s = 0.2$
- Constraints: $|x| \leq [15\ 10\ 15\ 10]'$, $|u_i| \leq 100$
- $T = 2, R_1 = R_2 = 1$

  $Q_1 = \mathrm{diag}(1, 1, 0, 0), Q_2 = \mathrm{diag}(0, 0, 1, 1)$



- $\lambda = 0.97$
- $\|x_k\|/\|x_0\|$ always below the **theoretical bound** $r_k = \lambda^k \sqrt{c_2/c_1}$

LEARNING AGENTS' COSTS FROM DATA

LEARNING PARAMETRIC MONOTONE GAMES

(Bemporad, Tatarenko, 2026)

- **Monotonicity** is to games what **convexity** is to optimization (existence/uniqueness of v-GNE, convergence of GNE-seeking algorithms)
- **Goal**: learn a **surrogate game** $\{J_i(x,p;\theta)\}_{i=1}^N$ that is **monotone** for all p
- **Direct learning** from agents' **cost samples** $\tilde{J}_{k,i} = \tilde{J}_i(x_k,p_k)$:

$$\mathcal{L}_J(\theta) = \frac{1}{KN}\sum_{k=1}^{K}\sum_{i=1}^{N}\left(J_i(x_k,p_k;\theta) - \tilde{J}_{k,i}\right)^2$$

- **Inverse learning** from **best-response samples** $x_{k,i}$:

$$\mathcal{L}_X(\theta) = \frac{1}{KN}\sum_{k=1}^{K}\sum_{i=1}^{N}\left\| \arg\min_{x_i} J_i(x_k,p_k;\theta) - x_{k,i}\right\|_2^2$$

- Shared constraints $x \in \mathcal{X}(p)$ added **after** training $\Rightarrow$ **monotone GNEP**

HOW TO ENFORCE MONOTONICITY?

- **Penalty**: penalize monotonicity violations, e.g.,

$$\mathcal{L}_M(\theta) = \frac{\gamma}{K}\sum_{k=1}^{K} \max\left\{0, \mu - \frac{\lambda_{\min}\left(\nabla_x F(x_k,p_k;\theta)+\nabla_x F'(x_k,p_k;\theta)\right)}{2}\right\}^2$$

but *no guarantee* that the learned game will be monotone for all p

- **By construction**: set $F = F_s + F_a$ with $\nabla F_s = \nabla F_s' \succeq 0$, $\nabla F_a = -\nabla F_a'$

$$\Rightarrow \quad J_i(x) = \Psi(x) + \sum_{j\neq i} x_i' S_{ij} x_j + r_i' x_i + \phi_i(x_{-i})$$

$\Psi(x)$ convex,
$\nabla^2\Psi(x) \succeq \mu I$

$S' = -S$

All monotone convex **quadratic** games, with $F(x) = Ax + q$, have

$A = C'C + D - D' + \mu I$, $\quad C$ upper triang., $\quad D$ block strictly upper triang.

- **Model**: take $C(p), D(p), q(p)$ = **NNs** of p, $\Psi(x,p)$ an **input-convex NN**, $\mu \geq 0$

EXAMPLE: INTERNET SWITCHING GAME

- Variant of the **internet switching** model (Facchinei, Kanzow, 2009)

$$\tilde{J}_i(x,p) = -\frac{x_i}{\sum_{j=1}^N x_j}\left(1 - p\sum_{j=1}^N x_j\right)$$

$$\sum_{j=1}^N x_j \le \frac{1}{p}, \quad x_i \ge 0.01, \quad p \in [0.5, 20]$$

- $N = 5$ agents, $K = 4000$ **cost samples** (direct learning), $\mu = 0$
- **Monotone quadratic model**, w/ or w/o **ICNN potential** Ψ (2 layers, 20 neurons)
- Comparison of **original vs learned** GNEP solutions under the same constraints:

potential Ψ	training time (s)	R^2 (%)	BR error	NE error
none	72.6	94.62	1.79e-02	3.60e-03
NN	300.7	95.40	1.43e-02	4.69e-03

R^2 = cost fit on test set, BR error on 200 test p's, NE error = $||\hat{x}^\star(p) - x^\star(p)||$ on 50 test p's

LEARNING GNE OF PARAMETRIC GAMES

LEARNING APPROXIMATE GNE SOLUTION MAPS

- **Goal**: learn offline $\hat{x}(p,\theta) \approx x^\star(p)$ for fast **online** evaluation (example application: **explicit** game-theoretic MPC)

- **Exact** explicit GNE maps exist for **LQ games**, but the number of regions can grow exponentially with the number of constraints (Hall, Bemporad, 2025)

- Regression on **GNE data** $\{p_k, x^\star(p_k)\}$ is problematic:
  - a GNEP must be **solved** for each sample p_k (expensive)
  - which GNE to pick if **multiple** GNEs exist?

- **Idea**: minimize the **Nikaido–Isoda** (NI) gap function (Nikaido, Isoda, 1955)

$$NI(x,p) = \sum_{i=1}^{N} \Big(J_i(x_i, x_{-i}, p) - \bar{J}_i(x_{-i}, p) \Big) \geq 0 \qquad NI(x^\star(p), p) = 0$$

$\bar{J}_i(x_{-i}, p) = J_i(\bar{x}_i(x_{-i}, p), x_{-i}, p)$ = best-response cost (=**value function**)

- **Value-function data** are much cheaper to obtain than GNE data!

NI-BASED LEARNING PROBLEM

(Bemporad, Tatarenko, 2026)

- Dataset $\{x_k, p_k, \bar{J}_k\}_{k=1}^M$, $\bar{J}_{k,i} = \bar{J}_i(x_{-i,k}, p_k)$
- **Step 1**: fit **value-function models** $\hat{J}_i(x_{-i}, p, \theta_{1i}) \approx \bar{J}_i(x_{-i}, p)$

$$\theta_{1i}^\star = \arg\min_{\theta_{1i}} \frac{1}{M} \sum_{k=1}^{M} \left(\hat{J}_i(x_{-i,k}, p_k, \theta_{1i}) - \bar{J}_{k,i}\right)^2 + r_{1i}(\theta_{1i})$$

- **Step 2**: train the **GNE model** $\hat{x}(p, \theta_2)$ (e.g., a feedforward NN)

$$\theta_2^\star = \arg\min_{\theta_2} \frac{1}{M} \sum_{k=1}^{M} \sum_{i=1}^{N} \ell\left(\hat{\nu}_i(p_k, \theta_2)\right) + \ell^c_{\beta,\gamma}(\theta_2) + r_2(\theta_2)$$

  - $\hat{\nu}_i(p_k, \theta_2) = J_i(\hat{x}(p_k, \theta_2), p_k) - \hat{J}_i(\hat{x}_{-i}(p_k, \theta_2), p_k, \theta_{1i}^\star)$ = approx. **NI term**
  - $\ell(\hat{\nu}_i) = \frac{1}{2}(\hat{\nu}_i + \sqrt{\hat{\nu}_i^2 + \epsilon}) \approx \max(\hat{\nu}_i, 0)$
  - $\ell^c_{\beta,\gamma}(\theta_2)$ = smooth max penalty of **shared-constraint** violations

EXAMPLE: NONLINEAR GNEP

- **Internet switching** model, reparameterized with $p \leftarrow 1/p$ (Facchinei, Kanzow, 2009)

$$J_i(x,p) = -\frac{x_i}{\sum_{j=1}^N x_j}\left(1 - \frac{\sum_{j=1}^N x_j}{p}\right)$$

$$\sum_{j=1}^N x_j \leq p, \quad x_i \geq 0.01$$

- $p \in [0.01N, 2], N = 2, \ldots, 10$ agents
- $1000(N-1)$ training samples
- $\hat{J}_i$: NN (10,5), swish
  $\hat{x}$: NN $(2N, 2N)$, ReLU
- **No constraint violation**, small best-response errors on 1000 **test** samples

N	$\bar{v}$, $v_{\max}$	$\bar{\epsilon}^2_{\rm BR}$	training time (s)
2	0	6.06e-06	17.1
3	0	9.11e-05	24.8
4	0	2.87e-05	51.4
5	0	9.52e-05	81.4
6	0	2.01e-04	124.4
7	0	9.72e-05	170.9
8	0	1.82e-04	234.5
9	0	3.34e-04	275.5
10	0	1.78e-03	348.2

$\bar{v}$, $v_{\max}$ = avg/max constraint violation

$\bar{\epsilon}^2_{\rm BR}$ = mean-squared best-response error

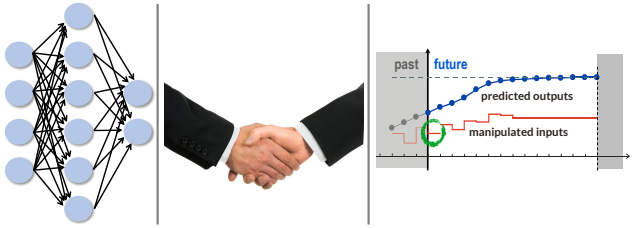
CPU: Apple M4 Max

Python library:

`github.com/bemporad/mpfit`

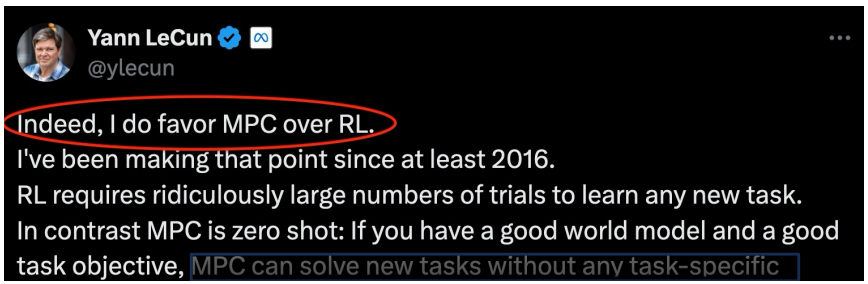
CONCLUSIONS

CONCLUSIONS



- **Learning** methods can greatly help **MPC design**:
  - **Models**: learn/adapt control-oriented models
  - **Costs**: tune MPC weights (e.g., from operators' preferences)
  - **Evaluation**: learn surrogates or explicit solution maps of the MPC law

- Will machine learning completely replace MPC?



(Yann LeCun, Twitter/X, August 25, 2024)

- Many open research problems at the crossroads of **learning**, **games**, and **MPC**!