EFFICIENT LEARNING ALGORITHMS FOR MODEL PREDICTIVE CONTROL

Alberto Bemporad

imt.lu/ab

Funded by the European Union

European Research Council
Established by the European Commission

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

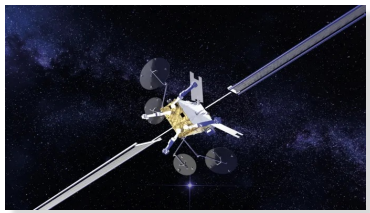
ODYS
Advanced Controls & Optimization

AI4I-CPS-IAS - Torino, September 16, 2025

INDUSTRIAL AUTOMATION AND CONTROL SYSTEMS MARKET


needpix.com


enel.com


thalesaleniaspace.com

Automotive sector
Aeronautics
Aerospace
Military sector
Manufacturing
Chemicals
Pharmaceuticals
Paper production
Mining & metals
Medical devices
Financial engineering
Electrical systems
Water resource management
Environmental systems
Logistics
...


airbus.com

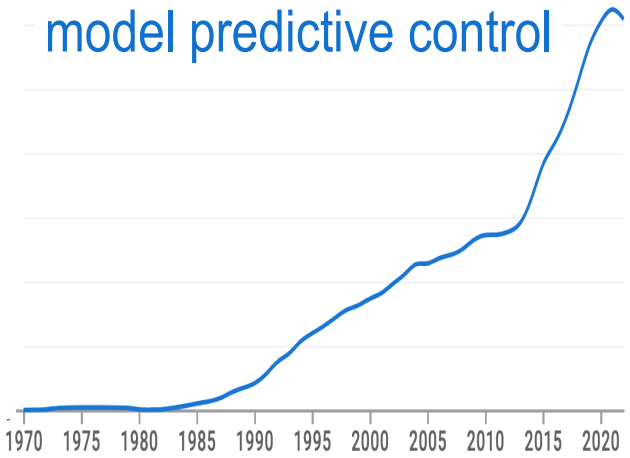

gm.com

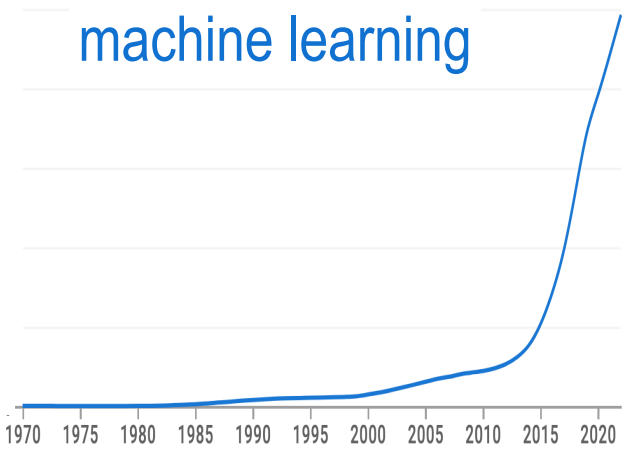

comau.com

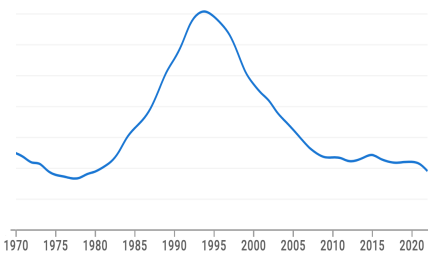
	2023/2024			Forecast		
World[1]	USD	206 bn	(2024)	USD	379 bn	(2030)
Italy[2]	USD	5 bn	(2023)	USD	11.4 bn	(2033)

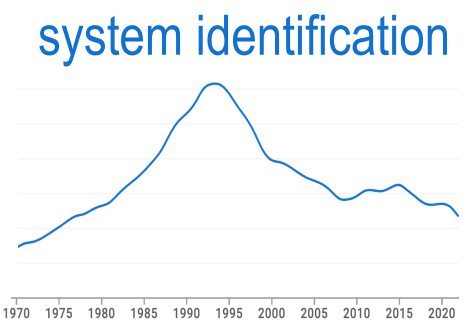
[1] grandviewresearch.com [2] apollorr.com

CONTROL SYSTEMS DESIGN METHODS





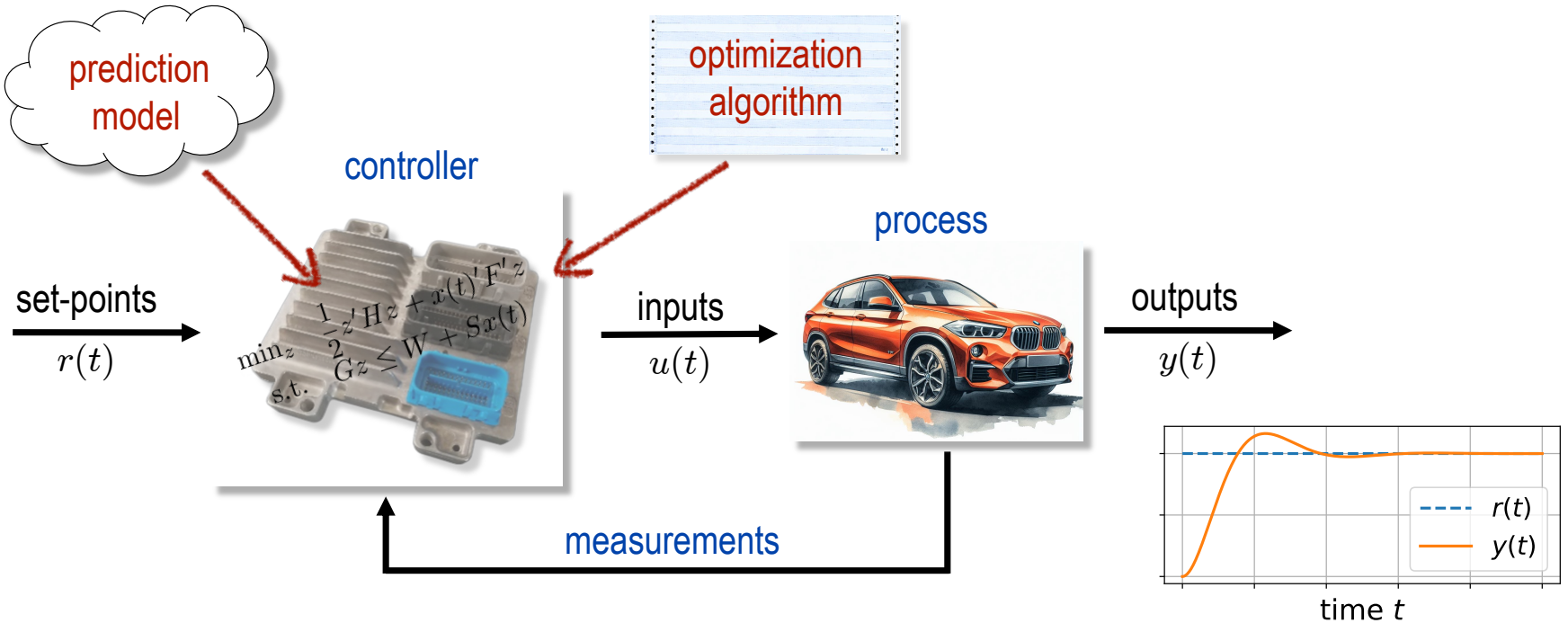




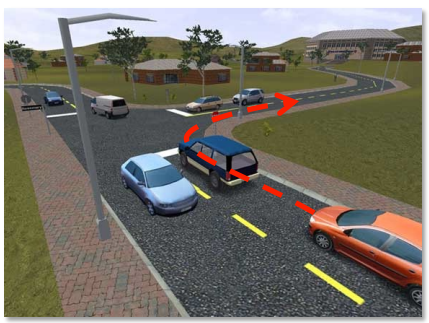
(source: https://books.google.com/ngrams)

MPC and **ML** = main R&D trends in industry for control systems design

MODEL PREDICTIVE CONTROL (MPC)



- **Main idea**: At each sample step, use a (simplified) dynamical **(M)odel** of the process to **(P)redict** its future evolution and choose the "best" **(C)ontrol** action accordingly



MODEL PREDICTIVE CONTROL

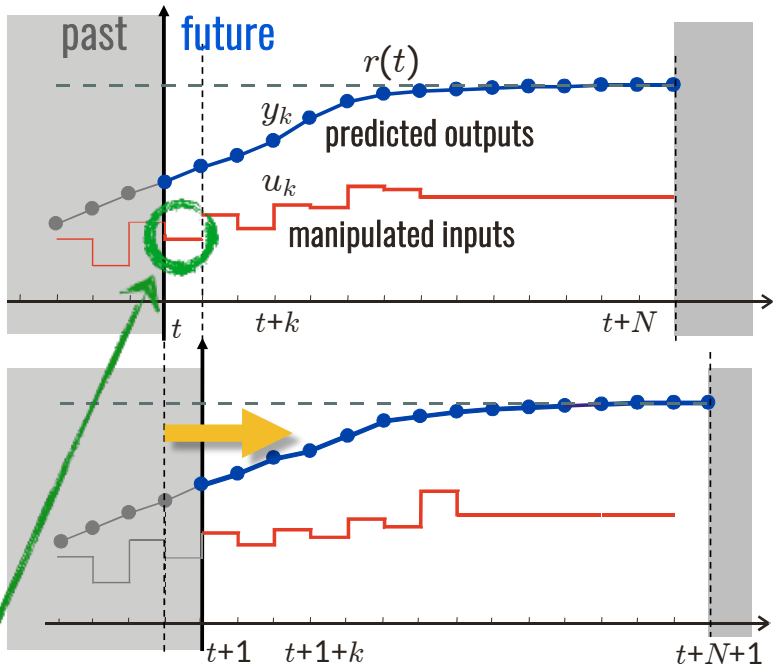
- MPC algorithm:

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

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

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

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



numerical optimization problem

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

Repeat at all time steps t

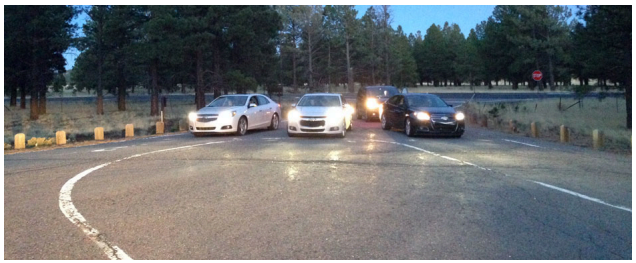
MPC IN INDUSTRY

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

- Used in the **process industries** since the 1980s (Qin, Badgewell, 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)



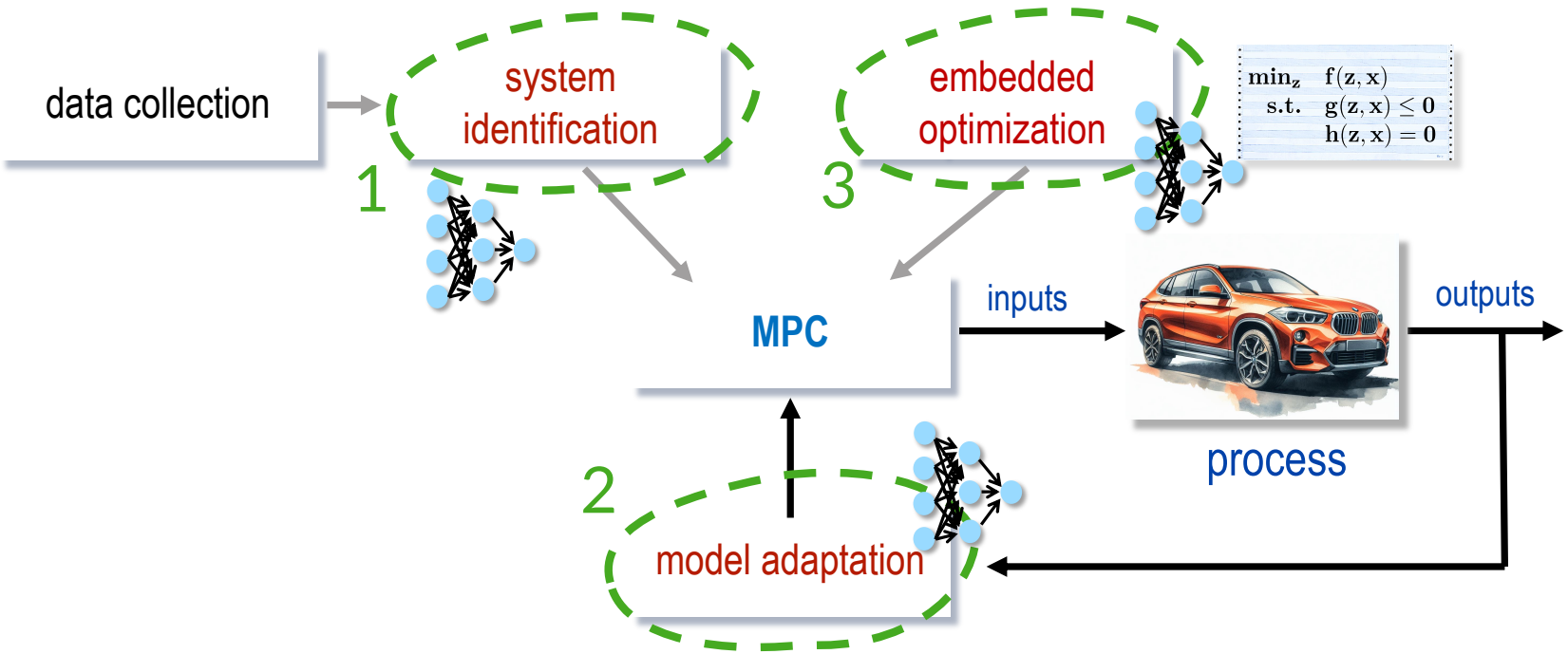
First known mass production of MPC in the automotive industry

and more are underway...

ODYS
Advanced Controls & Optimization

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

FOCUS OF MY TALK



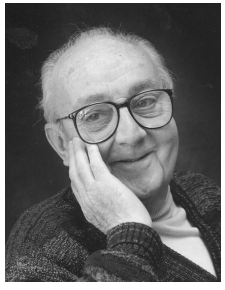
Focus of my talk:

1. Learn **nonlinear prediction models** from data
2. Learn **disturbance models** online for model adaptation
3. Learn **convex functions** for reducing online MPC computations

LEARNING CONTROL-ORIENTED NONLINEAR MODELS

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

(George E. P. Box)



CONTROL-ORIENTED MODELS

- A **complex model** implies a **complex MPC controller**
- We typically look for **small-scale models** (e.g., ≤ 10 states/inputs/outputs) with a **limited number of coefficients** (<1k params vs >300B of LLMs)
- **Limit nonlinearities** as much as possible (e.g., avoid very deep neural networks)
- Need to get the **best model** within a **poor model class** from a **rich dataset** (= limited risk of overfit)
- **Computation constraints**: solve the learning problem using limited resources (=our laptop, no supercomputing infrastructures)

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

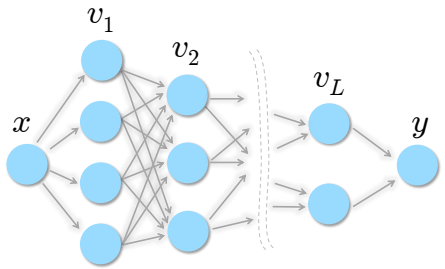
RECURRENT NEURAL NETWORKS

- **Neural networks** proposed for nonlinear system identification since the '90s

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

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

$$\begin{aligned} x_{k+1} &= f_x(x_k, u_k, \theta_x) \\ y_k &= f_y(x_k, \theta_y) \\ f_x, f_y &= \text{feedforward neural network} \end{aligned}$$



$$v_j = W_j f_{j-1}(v_{j-1}) + b_j$$

$$\theta = (W_1, b_1, \ldots, W_L, b_L)$$

(e.g.: general RNNs, LSTMs, RESNETS, physics-informed NNs, ...)

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

$$\min_{\substack{\theta_x, \theta_y \\ x_0, x_1, \ldots, x_{N-1}}} \quad r(x_0, \theta_x, \theta_y) + \frac{1}{N}\sum_{k=0}^{N-1} \ell(y_k, f_y(x_k, \theta_y))$$

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

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

GRADIENT DESCENT METHODS FOR TRAINING RNNS

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

$$\min_{\theta_x,\theta_y,x_0} r(x_0,\theta_x,\theta_y) + \frac{1}{N}\sum_{k=0}^{N-1} \ell(y_k, f_y(x_k,\theta_y)) = \min_{\theta_x,\theta_y,x_0} V(\theta_x,\theta_y,x_0)$$

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

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

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

(Kingma, Ba, 2015)

- **Main issue**: **slow convergence** (in theory and in practice)

GRADIENT DESCENT METHODS FOR TRAINING RNNS

- **Stochastic** gradient descent (SGD) can be even less efficient with RNNs:
  - collect a high 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 the **Hessian** $\nabla^2 V({\theta_x}^t, {\theta_y}^t, x_0^t)$

- **Quasi-Newton methods**: good tradeoff between convergence speed / solution quality and numerical complexity. Only requires the **gradient** $\nabla V({\theta_x}^t, {\theta_y}^t, x_0^t)$

TRAINING RECURRENT MODELS VIA L-BFGS

AI-generated image - DALL-E

SYSTEM IDENTIFICATION PROBLEM

- Class of nonlinear dynamical models (e.g., RNNs w/ linear bypass):

$$x_{k+1} = Ax_k + Bu_k + f_x(x_k, u_k; \theta_x)$$
$$\hat{y}_k = Cx_k + Du_k + f_y(x_k, u_k; \theta_y)$$

Special cases:

linear model, RNN, ...

- Loss function (open-loop prediction error + regularization)

$$\min_{z, x_1, \ldots, x_{N-1}} \; r(z) + \frac{1}{N} \sum_{k=0}^{N-1} \ell(y_k, Cx_k + Du_k + f_y(x_k, u_k; \theta_y))$$
$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k + f_x(x_k, u_k; \theta_x)$$
$$k = 0, \ldots, N-2$$

$$z = \begin{bmatrix} x_0 \\ \Theta \end{bmatrix}$$

$$\Theta = \begin{bmatrix} A(:) \\ B(:) \\ C(:) \\ D(:) \\ \theta_x \\ \theta_y \end{bmatrix}$$

- Condense the problem by eliminating the hidden states x_k and get

$$\min_z \; f(z) + r(z)$$

(nonconvex) nonlinear programming (NLP) problem

NLP PROBLEM

- If f and r **differentiable**: use any state-of-the-art unconstrained NLP solver, e.g., **L-BFGS** (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) (Liu, Nocedal, 1989)

- The gradient $\nabla f(z)$ can be computed efficiently by **automatic differentiation**

- However, **sparsifying** the model requires **non-smooth** regularizers:

$$r_1(z) = \tau \|z\|_1 \qquad r_g(z) = \tau_g \sum_{i=1}^{m} \|I_i z\|_2$$

ℓ_1-regularization — group-Lasso penalty

- **Group-Lasso penalties** can be used for reducing:
  - the **number of states**
  - the **number of inputs**
  - the **number of neurons** in each layer of f_x, f_y
  - ...

HANDLING NON-SMOOTH REGULARIZATION TERMS

(Bemporad, 2024)

1. If $r(x) = \sum_{i=1}^{n} r_i(x_i)$ and $r_i : \mathbb{R} \to \mathbb{R}$ is convex and positive semidefinite, the ℓ_1-regularized problem can be recast as a **bound-constrained NLP**:

$$\min_x f(x)+\tau\|x\|_1+r(x) \quad \Rightarrow \quad \min_{y,z\geq 0} f(y-z)+\tau[1\ \dots\ 1]\begin{bmatrix} y \\ z \end{bmatrix}+r(y)+r(-z)$$

$$x^* = y^* - z^*$$

Example: $r(x) = \|x\|_2^2$ then $r(y) + r(-z) = \left\|\begin{bmatrix} y \\ z \end{bmatrix}\right\|_2^2$ well-regularized augmented problem

2. If $r(x)$ is convex and symmetric wrt each component x_i and increasing for $x \geq 0$, and $\tau > 0$, then we can solve instead

$$\min_{y,z\geq 0} f(y-z)+\tau[1\ \dots\ 1]\begin{bmatrix} y \\ z \end{bmatrix}+r(y+z)$$

if $r(x)$ differentiable for $x \neq 0$ then $r(y+z)$ differentiable if any $y_i, z_j > 0$

Example: $r(x)$ = group-Lasso penalty + constraint $y, z \geq \epsilon$ = machine precision

JAX-SYSID LIBRARY

(Bemporad, 2024)

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

jax-sysid

```
import numpy as np
from jax_sysid.models import Model

def state_fcn(x,u,params):      state-update function, x(k+1)
    ...

def output_fcn(x,u,params):     output function, y(k)
    ...

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)
model.optimization(adam_epochs=1000, lbfgs_epochs=1000)
model.fit(Y, U)

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

```
pip install jax-sysid
```

github.com/bemporad/jax-sysid

QUASI-LPV MODEL IDENTIFICATION

- **Quasi-Linear Parameter Varying** (qLPV, a.k.a. "self-scheduled" LPV) models:

$$x_{k+1} = A(p_k)x_k + B(p_k)u_k$$
$$y_k = C(p_k)x_k + D(p_k)u_k$$

$$\begin{bmatrix} A(p_k) & B(p_k) \\ C(p_k) & D(p_k) \end{bmatrix} = \begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \sum_{i=1}^{n_p} \begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix} p_{ki}$$

where $p_k \in \mathbb{R}^{n_p}$ is the **scheduling parameter** vector, such as

$$p_{ki} = \frac{1}{1 + e^{-f(x_k, u_k; \theta_i)}}, \; i = 1, \ldots, n_p - 1$$

where $f(x_k, u_k; \theta_i)$ is a FNN with linear output layer and parameters θ_i

- qLPV models are a very powerful class of control-oriented nonlinear models

```
from jax_sysid.models import qLPVModel

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

EXAMPLE: QLPV MODEL IDENTIFICATION

- Generate 5000 training data and 1000 test data from the NL dynamics

$$x_{k+1} = \begin{bmatrix} 0.5\sin(x_{1k}) + 1.7\cos(0.5x_{2k})u_k \\ 0.6\sin(x_{1k}+x_{3k}) + 0.4\,\mathrm{atan}(x_{1k}+x_{2k}) \\ 0.4\,e^{-x_{2k}} + 0.9\sin(-0.5x_{1k})u_k \end{bmatrix} + \xi_k$$

$$y_k = \mathrm{atan}(2.2x_{1k}^3) + \mathrm{atan}(1.8x_{2k}^3) + \mathrm{atan}(-x_{3k}^3) + z_k$$

where $\xi_k, z_k \in \mathcal{N}(0, 0.01^2)$ and u_k uniformly generated in $[-\frac{1}{2}, \frac{1}{2}]$

- p_k = 2-layer FNN (6 neurons each) + swish activation + sigmoid output function

- Training results:
  - 1000 Adam + 5000 L-BFGS iters
  - CPU time measured on [Apple M4 Max]

model	n_p	Best Fit Rate training data	Best Fit Rate test data	CPU time (s)
LTI	0	71.35	71.36	1.3
qLPV	1	93.57	93.55	20.1
qLPV	2	95.54	95.51	22.6
qLPV	3	96.04	95.94	26.4

COMBINED LEARNING OF MODEL AND INVARIANT SETS

(Mulagaleti, Bemporad, 2025)

- **Goal**: learn a model for **control design** under **constraints** $y \in \mathcal{Y}, u \in \mathcal{U}$
- **Key idea**: add **regularization term** $r(\theta)$ in training problem (θ = model coeffs):

$$
\begin{aligned}
r(\theta) = \quad & \min_{R} \text{conservativeness}(R) \\
\text{s.t.} \quad & C \cdot R \oplus W \subseteq \mathcal{Y} \\
& \forall x_t \in R, \exists u_t \in \mathcal{U} : \; x_{t+1} \in R
\end{aligned}
$$

- $r(\theta) < \infty \Rightarrow \exists$ **control invariant set** R
- small $r(\theta) \Rightarrow$ less conservative R
- qLPV + polytopic sets $\Rightarrow r(\theta)$ differentiable

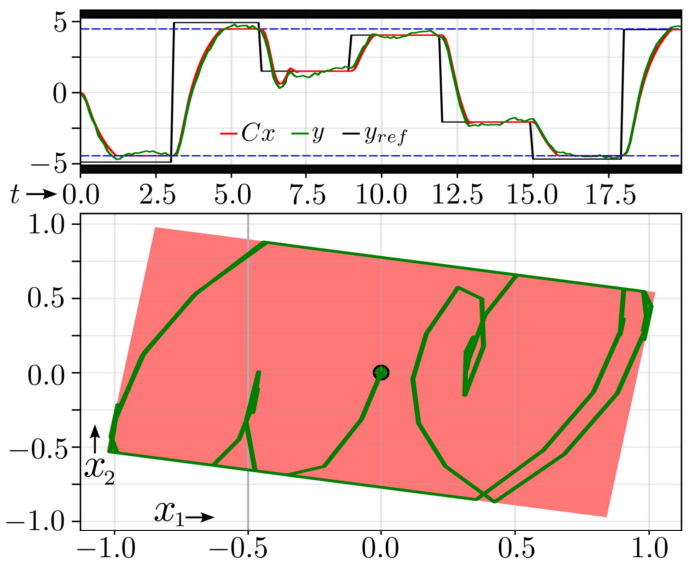
- **Example**: $1.5\dddot{y} + \ddot{y} + \dot{y} + y + 1000y^3 = u$

10,000 training + 2,000 disturbance-set estimation data

learned model: $n_x = 4$ states, $n_p = 6$ scheduling params

p-function = shallow FNN with 3 neurons

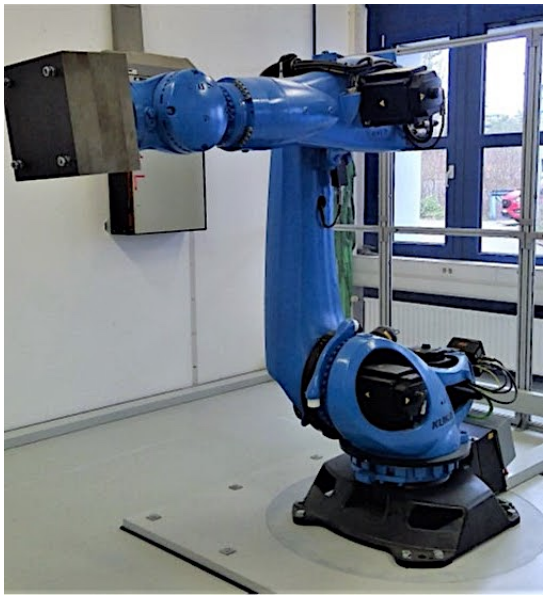
$r(\theta)$ almost does not perturb quality of fit



INDUSTRIAL ROBOT BENCHMARK

(Weigand, Götz, Ulmen, Ruskowski, 2022)

- KUKA KR300 R2500 ultra SE industrial robot, full robot movement



nonlinearbenchmark.org

- **6 inputs** (torques), **6 outputs** (joint angles), w/ backlash, highly **nonlinear** and coupled, **slightly oversampled** ($\|y_k - y_{k-1}\|$ is often very small)

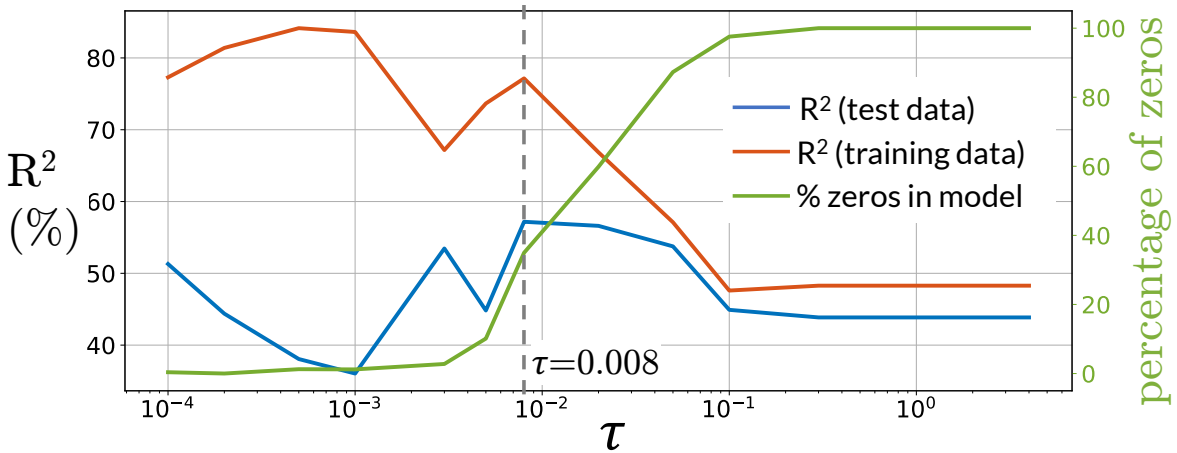
- Identification benchmark dataset (forward model):
  - Sample time: $T_s = 100$ ms
  - N = 39988 training samples
  - N_{test} = 3636 test samples

- Very challenging NL-SYSID benchmark on **`nonlinearbenchmark.org`**

INDUSTRIAL ROBOT BENCHMARK: RESULTS

(Bemporad, 2024)

- State $x \in \mathbb{R}^{12}$, f_x, f_y with **36** and **24** neurons, **swish** activation fcn $\frac{x}{1+e^{-x}}$
- Total number of training parameters: $\dim(\theta_x) + \dim(\theta_y) = 1590$



(best R^2 in 30 runs)

- Model quality measured by **average R^2-score** on all outputs:

$$R^2 = \frac{1}{n_y}\sum_{i=1}^{n_y} 100\left(1 - \frac{\sum_{k=1}^{N}(y_{k,i} - \hat{y}_{k,i|0})^2}{\sum_{k=1}^{N}(y_{k,i} - \frac{1}{N}\sum_{i=1}^{N} y_{k,i})^2}\right)$$

- Training time $\approx$ 12 min per run on a single core [Apple M1 Max]
  (3192 variables, 2000 Adam iterations + 2000 L-BFGS-B iterations)

INDUSTRIAL ROBOT BENCHMARK: RESULTS

- **Open-loop simulation** errors ($\rho = 0.01, \tau = 0.008$):

	R^2 (training) RNN model	R^2 (test) RNN model	R^2 (training) linear model	R^2 (test) linear model	
average	77.1493	57.1784	48.2789	43.8573	**jax-sysid**
			39.2822	32.0410	**n4sid**

- More parameters/smaller regularization leads to overfitting training data

- **Pure Adam** vs **LBFG-B+Adam** vs **OWL-QN** (Andrew, Gao, 2007): ($\tau = 0.008$)

solver	adam iters	fcn evals	$\overline{R^2}$ training	$\overline{R^2}$ test	# zeros (θ_x, θ_y)	CPU time (s)
L-BFGS-B	2000	2000	**77.1493**	**57.1784**	556/1590	**309.87**
OWL-QN	2000	2000	74.7816	54.0531	**736/1590**	449.17
Adam	6000	0	71.0687	54.3636	1/1590	389.39

- Adam is unable to sparsify the model

ONLINE TRAINING VIA EXTENDED KALMAN FILTERING

TRAINING RNNS BY EKF

(Puskorius, Feldkamp, 1994) (Wang, Huang, 2011) (Bemporad, 2023)

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

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

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

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

The model θ_x, θ_y can be learned offline by processing a given dataset multiple times, and also **adapted on line** from streaming data (u_k, y_k)

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

TRAINING RNNS BY EKF

(Bemporad, 2023)

- EKF can be generalized to handle **general strongly convex and smooth** loss functions $\ell(y_k, \hat{y}_k)$

- Strongly convex smooth regularization terms $r(x_0, \theta_x, \theta_y)$ can be handled similarly

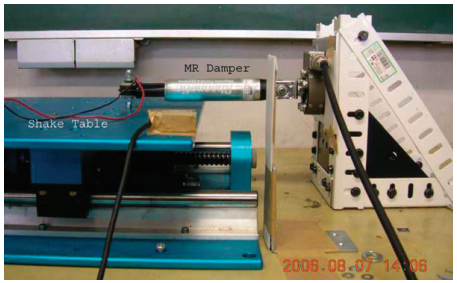
- Can handle **ℓ_1-penalties** $\lambda \left\| \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} \right\|_1$, useful to **sparsify** θ_x, θ_y by changing the EKF update into

$$\begin{bmatrix} \hat{x}(k|k) \\ \theta_x(k|k) \\ \theta_y(k|k) \end{bmatrix} = \begin{bmatrix} \hat{x}(k|k-1) \\ \theta_x(k|k-1) \\ \theta_y(k|k-1) \end{bmatrix} + M(k)e(k) - \lambda P(k|k-1) \begin{bmatrix} 0 \\ \text{sign}(\theta_x(k|k-1)) \\ \text{sign}(\theta_y(k|k-1)) \end{bmatrix}$$

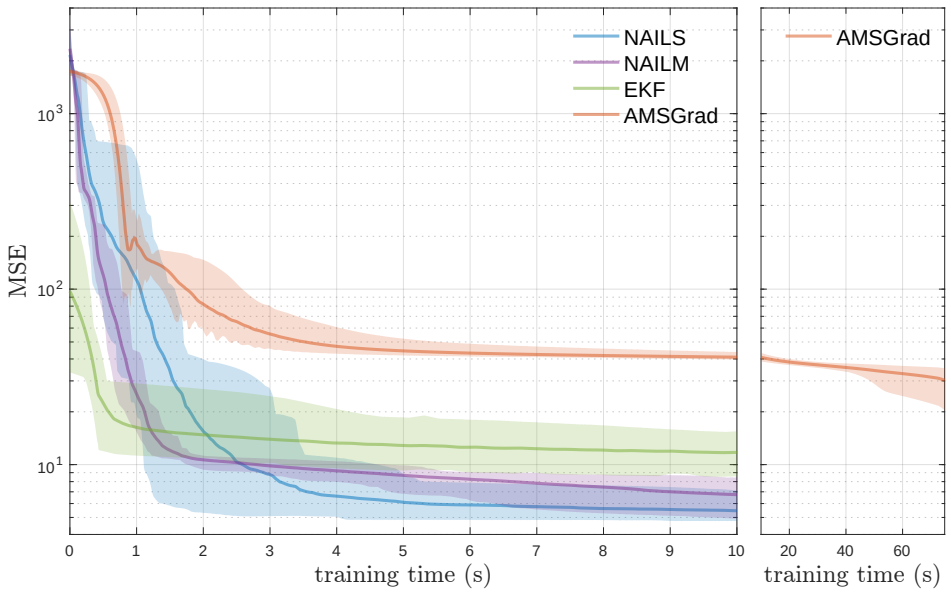
TRAINING RNNS BY EKF - EXAMPLE

- **Example**: **magneto-rheological fluid damper**
  N=2000 data used for training, 1499 for testing the model

  (Wang, Sano, Chen, Huang, 2009)



- RNN model: 4 states, shallow NNs w/ **4 neurons**, **I/O feedthrough**



MSE loss on training data, mean value (std) over 20 runs from different random initial weights

$$\text{BFR} = 100\left(1 - \frac{\|Y-\hat{Y}\|_2}{\|Y-\bar{y}\|_2}\right)$$

BFR (Best Fit Rate)	training	test
NAILS	**94.41** (0.27)	89.35 (2.63)
NAILM	94.07 (0.38)	89.64 (2.30)
AMSGrad	84.69 (0.15)	80.56 (0.18)
EKF	91.41 (0.70)	87.17 (3.06)
L-BFGS	93.19 (0.46)	90.33 (0.46)

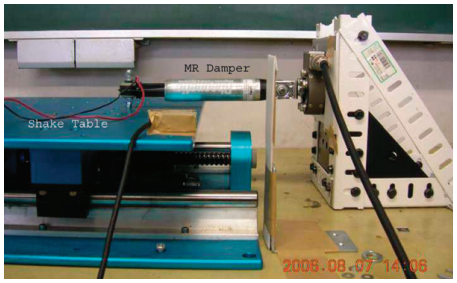
NAILS = GNN method with **line search**

NAILM = GNN method with **LM steps**

(Bemporad, 2023)

TRAINING RNNS BY EKF – EXAMPLE

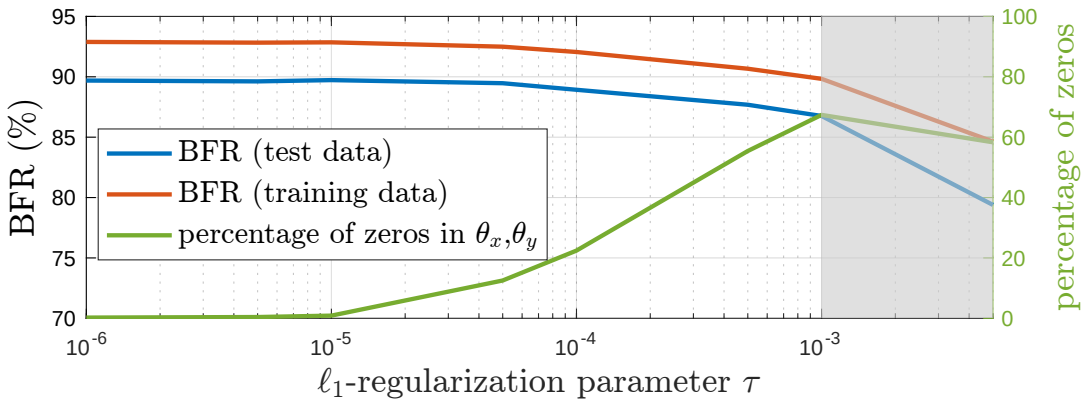
- **RNN model**: **4** states, shallow NNs with **6 neurons** each, **atan** activation, I/O feedthrough



- Compare BFR wrt NNARX model (SYS-ID TBX):

  EKF = **92.82**, Adam = **89.12**, NNARX(6,2) = **88.18** (**training**)
  EKF = **89.78**, Adam = **85.51**, NNARX(6,2) = **85.15** (**test**)

- Repeat EKF-based training with ℓ_1-penalty $\tau \left\| \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} \right\|_1$



- Main advantage of EKF: **online learning** possible!

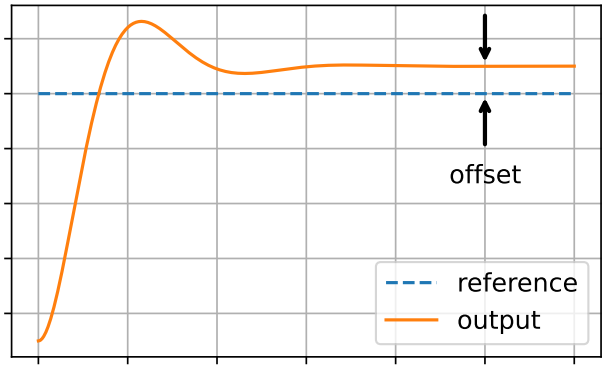
MODEL ADAPTATION AND OFFSET-FREE TRACKING

OUTPUT INTEGRATORS AND OFFSET-FREE TRACKING

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

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

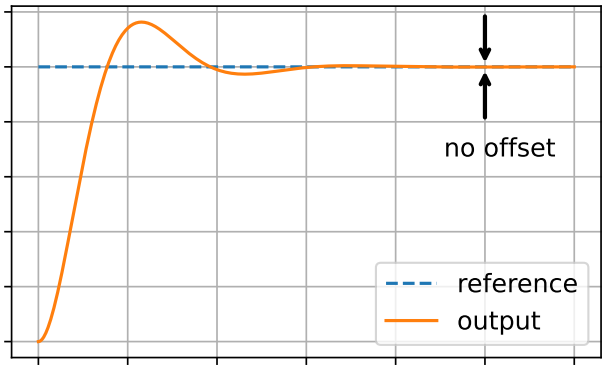
prediction model $\neq$ plant dynamics



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

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

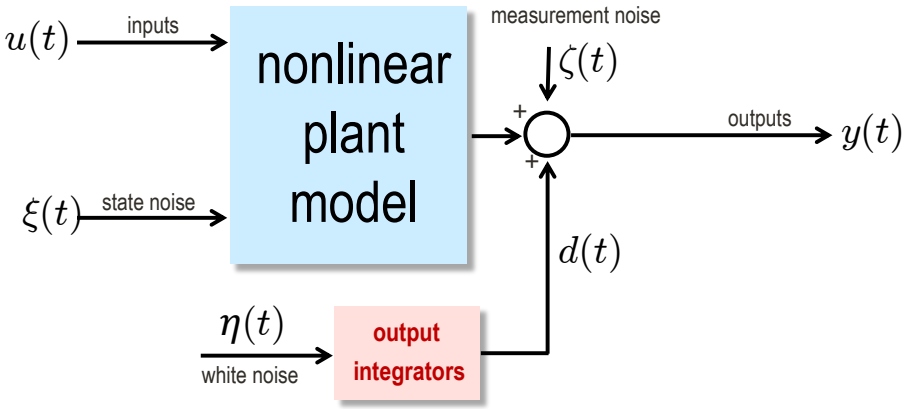
augmented prediction model



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

(Pannocchia, Gabiccini, Artoni, 2015)

OUTPUT INTEGRATORS AND OFFSET-FREE TRACKING



$$\begin{cases} x_{k+1} &= f(x_k, u_k) + \xi_k \\ d_{k+1} &= d_k + \eta_k \\ y_k &= g(x_k) + d_k + \zeta_k \end{cases}$$

ξ_k, η_k, ζ_k are noise terms

- Use an **extended Kalman filter** to estimate $\hat{x}(t)$ and $\hat{d}(t)$ from $y(t)$

- Intuitively, we get offset-free tracking in steady state because:
  - the observer makes $g(\hat{x}(t)) + \hat{d}(t) \to y(t)$ **(estimation error)**
  - the MPC controller makes $g(\hat{x}(t)) + \hat{d}(t) \to r(t)$ **(predicted tracking error)**
  - the combination of the two makes $y(t) \to r(t)$ **(actual tracking error)**

- In steady state, the term $\hat{d}(t)$ compensates for model mismatch (DC-gain error)

NONLINEAR DISTURBANCE MODELS FOR MPC

(Krupa, Zanon, Bemporad, 2025)

- Generalize the prediction model with nonlinear disturbance models:

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

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

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

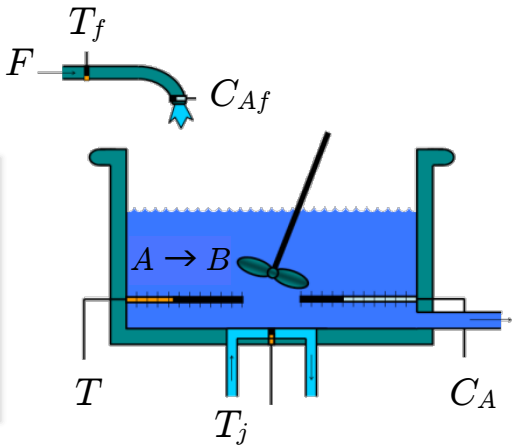
- **Motivation**: training the full model online may be difficult (lack of excitation, trustworthiness of the model, computational demand, etc.)

- Under certain assumptions, we can show that the tracking error $y(k) - r(k)$ **converges asymptotically to zero**, **even if $r(k)$ is not constant**

EXAMPLE: CSTR PROCESS

- MPC control of a diabatic **continuous stirred tank reactor (CSTR)**
- Process model is nonlinear (Bequette, 1998)

$$\begin{aligned}\frac{dC_A}{dt} &= \frac{F}{V}(C_{Af} - C_A) - C_A k_0 e^{-\frac{\Delta E}{RT}} \\ \frac{dT}{dt} &= \frac{F}{V}(T_f - T) + \frac{UA}{\rho C_p V}(T_j - T) - \frac{\Delta H}{\rho C_p} C_A k_0 e^{-\frac{\Delta E}{RT}}\end{aligned}$$

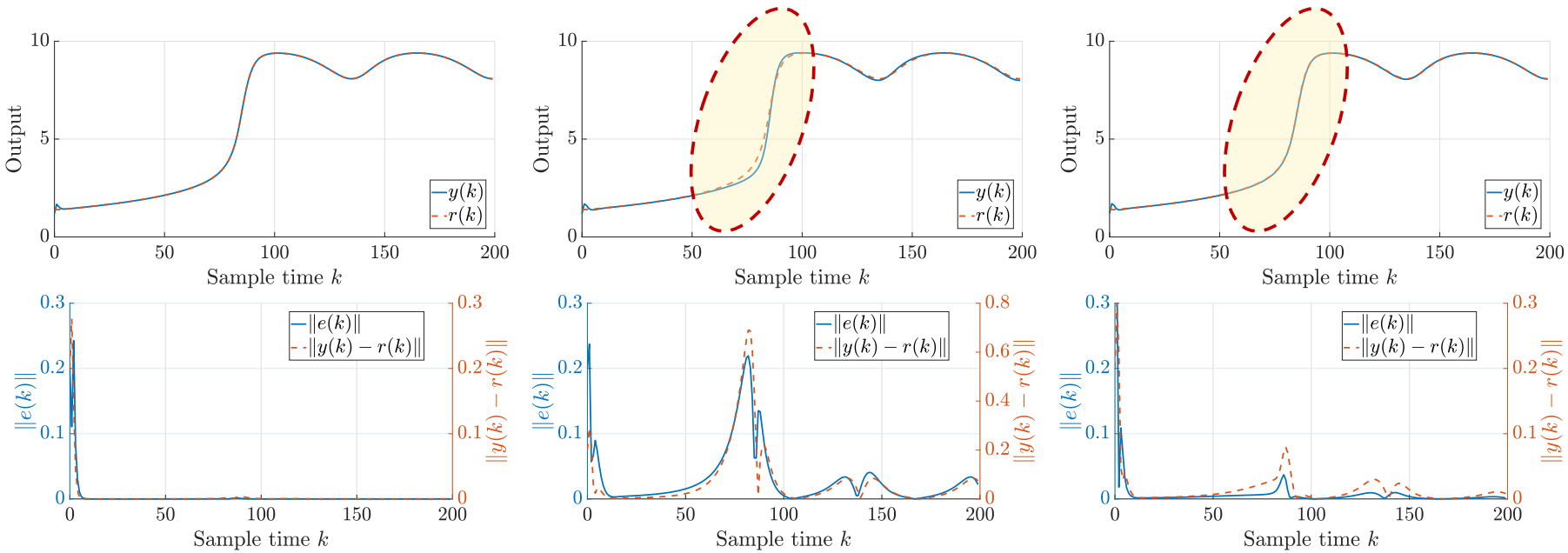


  - T : temperature inside the reactor $[K]$ (state)
  - C_A : concentration of the reactant in the reactor $[kgmol/m^3]$ (state)
  - T_j : jacket temperature $[K]$ (input)
  - T_f : feedstream temperature $[K]$ (measured disturbance)
  - C_{Af} : feedstream concentration $[kgmol/m^3]$ (measured disturbance)

- Objective: **manipulate** T_j to **regulate** C_A on desired setpoint

EXAMPLE: CSTR PROCESS

- Generic trackable reference signal $r(k)$ (**w/ preview**)



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

- The constant disturbance model is worse than FNN disturbance model, especially when $r(k)$ changes rapidly

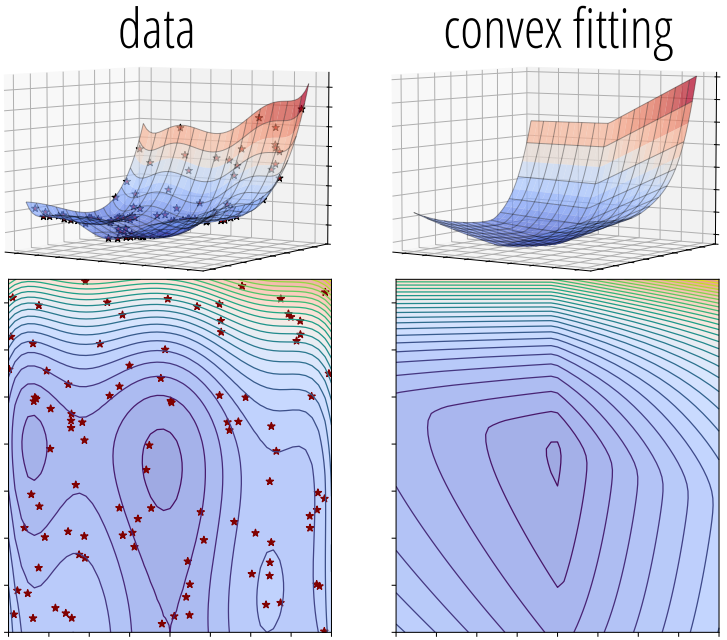
LEARNING CONVEX FUNCTIONS

LEARNING PARAMETRIC CONVEX FUNCTIONS

- **Goal**: learn a **parametric convex function** (PCF) from data (x_k, θ_k, y_k)

$$y = f(x, \theta) \qquad f : \mathbb{R}^n \times \Theta \to \mathbb{R}^d$$

with $f(x, \theta)$ **convex** wrt the **variable** $x \in \mathbb{R}^n$ for each **parameter** $\theta \in \Theta$



- **Use**: **optimize** f wrt x for each given θ in production

- **Example**: $f(x, \theta) = \frac{1}{2} x' F'(\theta) F(\theta) x + c(\theta) x + h(\theta)$

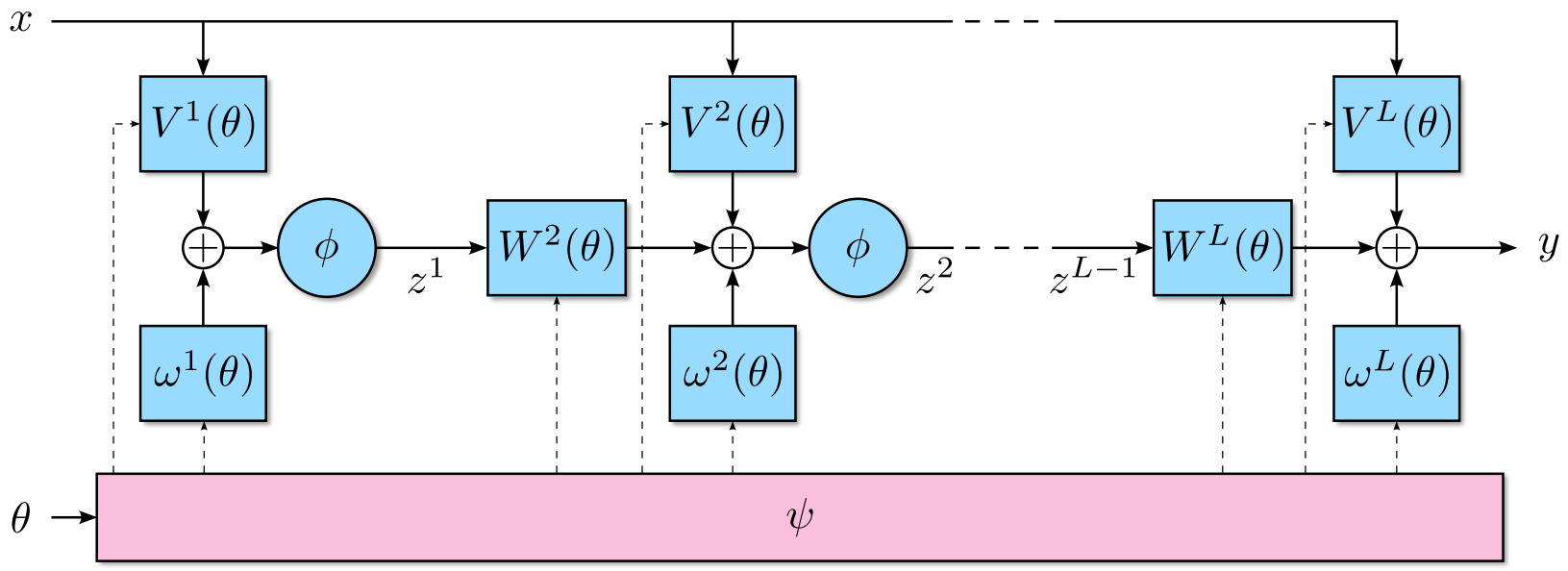
- Several **input-convex** NN architectures have been proposed in the literature

(Amos, Xu, Kolter, 2017) (Calafiore, Gaubert, Possieri, 2020)

NEURAL PCF ARCHITECTURE

(Schaller, Bemporad, Boyd, 2025)

- f = neural network with weights $V^i(\theta), W^i(\theta)$, biases $\omega^i(\theta)$, and activation ϕ



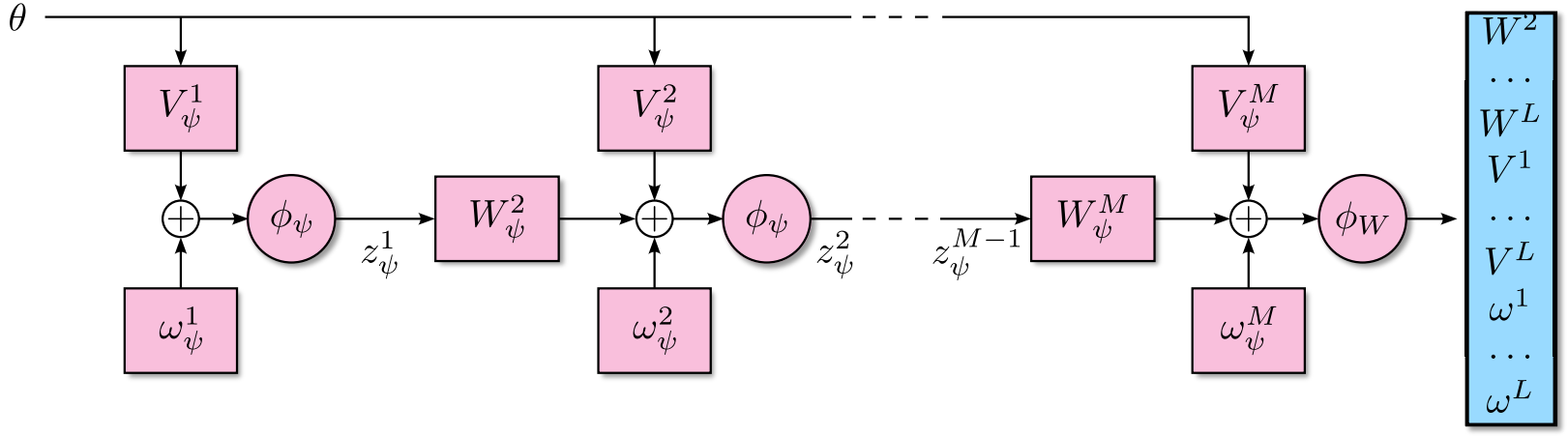
- V^i, W^i, ω^i generated by another network ψ (to be learned)
- Activation function ϕ is **nondecreasing** and **convex** (e.g., ReLU, softplus)
- The weights $W^i(\theta)$ are **elementwise nonnegative** for all θ

→ $f(x,\theta)$ **convex for all** θ

NEURAL PCF ARCHITECTURE

- The NN ψ is nonlinear re-parametrization from θ to the PCF weights

- ψ has weights $w = (W_\psi^2, \ldots, W_\psi^M, V_\psi^1, \ldots, V_\psi^M, \omega_\psi^1, \ldots, \omega_\psi^M)$



- The last layer of ψ makes $W^i(\theta)$ elementwise nonnegative $\forall \theta$

Examples:

$$W^i(\theta) = \max(V_\psi^M \theta + W_\psi^M z_\psi^{M-1} + \omega_\psi^M, 0)$$

$$W^i(\theta) = (V_\psi^M \theta + W_\psi^M z_\psi^{M-1} + \omega_\psi^M)^2$$

THE LPCF PACKAGE

- Open-source package for fitting a PCF to given data (Schaller, Bemporad, Boyd, 2025)

```
pip install lpcf
```

https://github.com/cvxgrp/lpcf

- Customizable neural network architecture

- Customizable loss and regularization

- Relies on **`jax_sysid`** (Adam + L-BFGS) for training

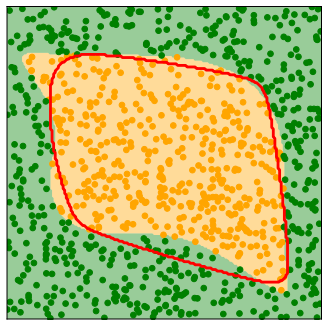
- Returns the PCF f as
  - a **JAX** function for fast evaluation (and differentiation)
  - a **CVXPY** expression for use in optimization models (Diamond, Boyd, 2016)

USING THE LPCF PACKAGE

```
from lpcf.pcf import PCF

# observed data
Y = ... # shape (N, d)
X = ... # shape (N, n)
Theta = ... # shape (N, p)

# fit PCF to data
pcf = PCF()
pcf.fit(Y, X, Theta)

# export PCF to CVXPY
x = cp.Variable((n, 1))
theta = cp.Parameter((p, 1))
y = pcf.tocvxpy(x, theta) # CVXPY expression
prob = cp.Problem(cp.Minimize(y + ...))
...

f = pcf.tojax() # JAX function f(x, θ)
```

Additional features:

- add (convex) **quadratic term** to the neural network
- require f to be **monotone** in x
- require f to be **nonnegative**
- require $\arg\min_x f(x,\theta) = g(\theta)$ for a given function g
- fit a parametrized convex set $C(\theta) = \{x \mid f(x,\theta) \leq 0\}$ (**convex classification** problem)



EXAMPLE: APPROXIMATE DYNAMIC PROGRAMMING (ADP)

- Consider the **input-affine** nonlinear system

$$x_{t+1} = F(x_t, \theta) + G(x_t, \theta)u_t, \quad t = 0, 1, \dots$$

- θ are measured parameters (e.g., physical quantities)
- **Goal**: for each given initial state x_0, find $u_0, u_1, \dots$ that minimize

$$J(x_0) = \sum_{t=0}^{\infty} H(x_t, u_t, \theta) = H(x_0, u_0, \theta) + \underbrace{\sum_{t=1}^{\infty} H(x_t, u_t, \theta)}_{\text{cost to go}}$$

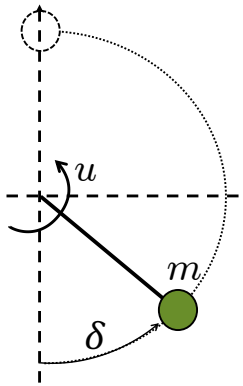
- **ADP controller** (i.e., MPC with horizon $N = 1$):

$$u_t = \arg\min_u \left(H(x_t, u, \theta) + f(F(x_t, \theta) + G(x_t, \theta)u, \theta) \right), \quad t = 0, 1, \dots$$

- **Convex problem** if $f(x, \theta)$ = PCF approx of $y = J(x, \theta)$ and H convex in u

EXAMPLE: APPROXIMATE DYNAMIC PROGRAMMING (ADP)

- **Example**: swing up inverted pendulum
  $x = [\delta, \dot{\delta}]', \theta = m > 0$ (mass)

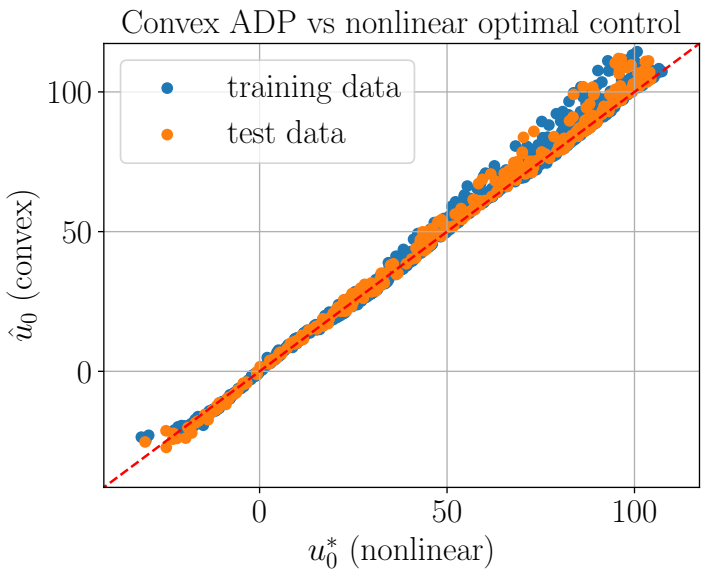


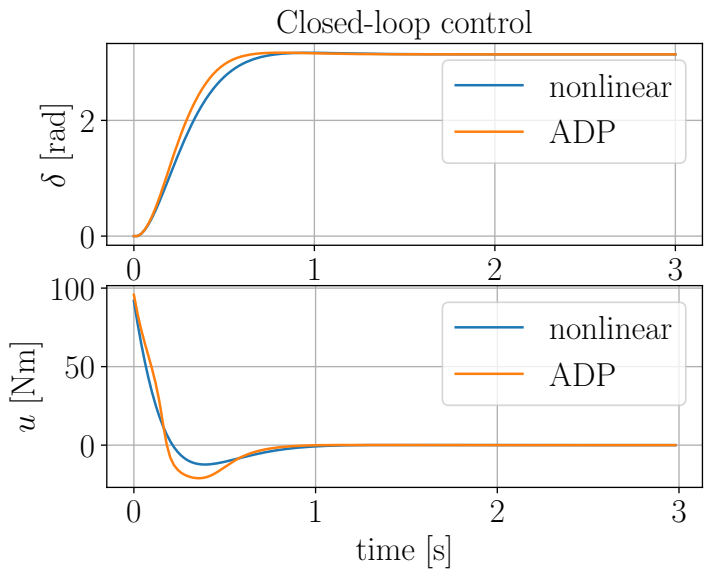
- Solve nonlinear optimal control problem

$$J(x_0) = \sum_{t=0}^{150} H(x_t, u_t, \theta)$$

on 1000 data points $(x_k, \theta_k), \theta_k \in [0.5, 2], \delta_k \in [-\pi/6, 7\pi/6], \dot{\delta}_k \in [-1, 1]$

- Fit PCF $f(x, \theta)$ and use CVXPY to solve the ADP problem online

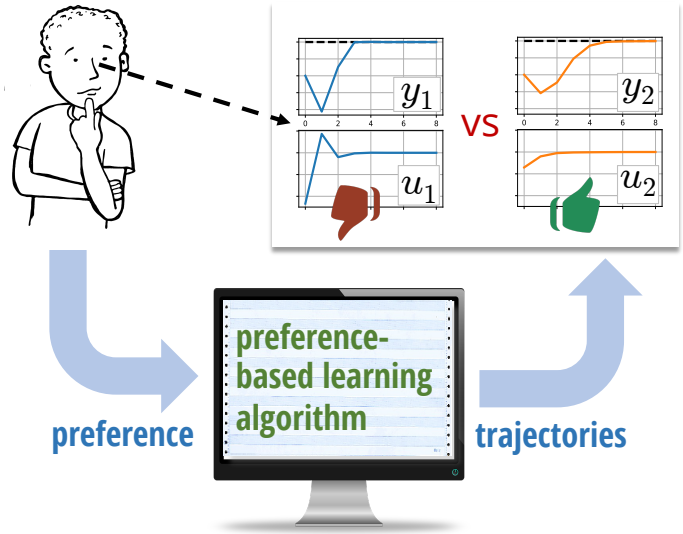




LEARNING MPC FROM PREFERENCES

LEARNING MPC CONTROLLER FROM PREFERENCES

- **Problem**: how to define the **MPC cost function** to minimize at runtime?
  - A clear KPI (key performance indicator) to optimize may not be available (**no KPI** or **multiple KPIs**)
  - A quantitative criterion leaves no room for **qualitative** assessments by a human calibrator



- **Approach**: **preference-based optimization** over **control policy parameters** via Bayesian optimization (Brochu, de Freitas, Ghosh, 2007) or radial basis functions (**GLISp**) (Bemporad, Piga, 2021) (Zhu, Piga, Bemporad, 2022)

```
pip install glis
```

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

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

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

PREFERENCE-BASED LEARNING OF MPC COST FUNCTION

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

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

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

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

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

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

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

$$\hat{\pi}(T_i, T_j, \theta) = \frac{1}{1 + \exp\left(\hat{\sigma}(T_i, \theta) - \hat{\sigma}(T_j, \theta)\right)} \qquad \hat{\sigma}(T, \theta) = \mathbf{PCF}$$

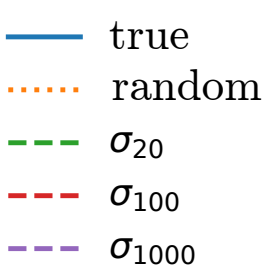
PREFERENCE-BASED LEARNING OF MPC COST FUNCTION

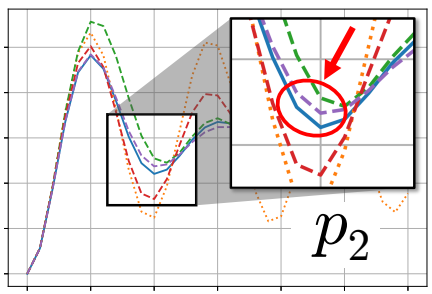
- **Learned MPC controller**: at each time t, given $x(t)$, solve the **convex problem**

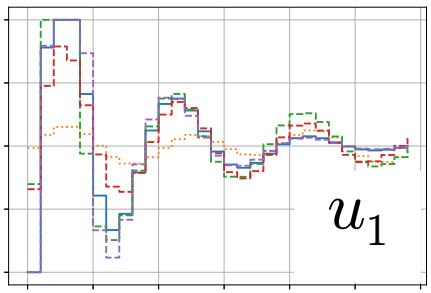
$$
\begin{aligned}
T_t \in \arg\min_T \ & \sigma(T, \theta^*) \\
\text{s.t. } & x_{k+1} = Ax_k + Bu_k, \quad k = 0, \dots, N-1 \\
& x_0 = x(t), \quad u_k \in \mathcal{U}, \quad x_k \in \mathcal{X} \\
& T = (X, U), \quad X = [x_0, \dots, x_N], \quad U = [u_0, \dots, u_{N-1}]
\end{aligned}
$$

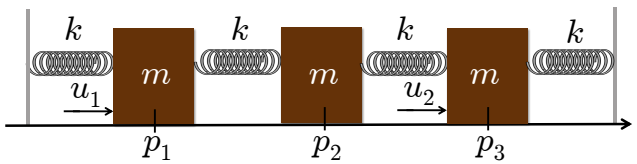
and apply $u(t)$ = 1st control move in T_t [we assume $\mathcal{U}, \mathcal{X}$ are convex sets]

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







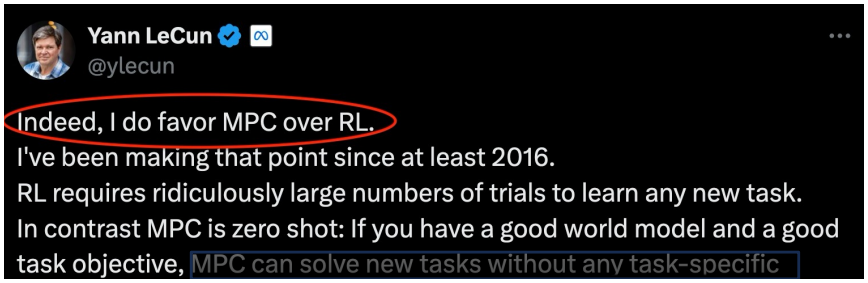


Training datasets: 20, 100, 1000 comparisons generated from 50 trajectories

CONCLUSIONS

CONCLUSIONS

- **ML** very useful to get **control-oriented models** and **control laws** from **data**
- **ML** cannot replace control engineering:
  - Blindly applying **deep learning** can lead to useless models for embedded control
  - **Model-based** MPC design is more sample-efficient, and performs tasks it wasn't trained for, better than **model-free** **reinforcement learning**



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

- **Some current research topics**:
  - How to get good-quality training data (**active learning**) (Xie, Bemporad, *CDC*, 2024)
  - More efficient methods for **non-smooth nonlinear optimization** with constraints

    (Adeoye, Latafat, Bemporad, 2025)

ERC Advanced Grant "COMPACT" (2024-2029)

Funded by the European Union

European Research Council
Established by the European Commission