LEARNING-BASED METHODS FOR MODEL PREDICTIVE CONTROL

Alberto Bemporad

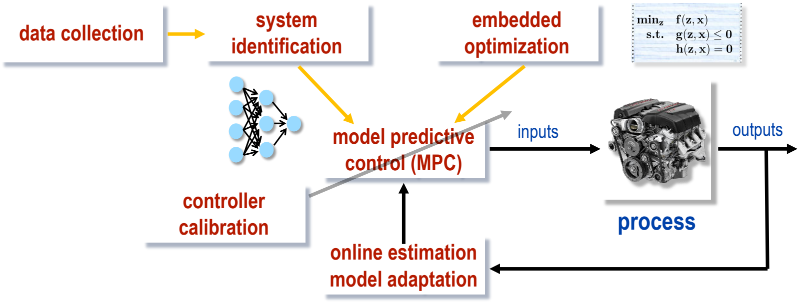
imt.lu/ab

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

ODYS Advanced Controls & Optimization

April 18, 2023

RESEARCH ISSUES IN EMBEDDED MPC DESIGN



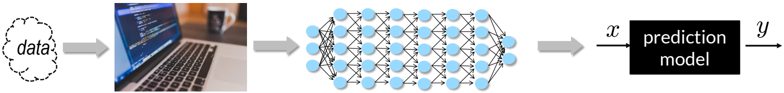
Focus of my talk:

- How to learn **nonlinear** and **piecewise affine models** from data
- How to **adapt** model parameters and **estimate** hidden model states
- How to speedup the **calibration** and **approximation** of the MPC law

LEARNING PREDICTION MODELS FOR MPC

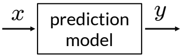
CONTROL-ORIENTED NONLINEAR MODELS

- **Black-box** models: purely data-driven. Use training data to fit a prediction model that can explain them (**need good data to get a good model**)



- **Physics-based** models: use physical principles to create a prediction model (**fewer parameters to learn, better generalizes on unseen data**)

$$\begin{aligned}
\dot{p}_1 &= k_1(W_c + W_{egr} - k_e p_1) + \frac{\dot{T}_1}{T_1} p_1\\
\dot{p}_2 &= k_2(k_e p_1 - W_{egr} - W_t + W_f) + \frac{\dot{T}_2}{T_2} p_2\\
\dot{P}_c &= \frac{1}{\tau}(P_c - \eta_m P_t)
\end{aligned}$$



- **Gray-box** (or **physics-informed**) models: mix of the two, can be quite effective

> "All models are wrong, but some are useful."
>
> (George E. P. Box)

NONLINEAR SYS-ID BASED ON NEURAL NETWORKS

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

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

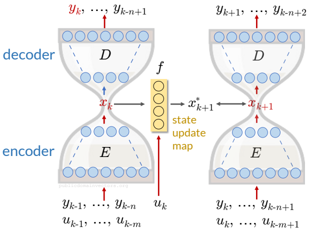
- **NNARX** models: use a **feedforward neural network** to approximate the nonlinear difference equation $y_t \approx \mathcal{N}(y_{t-1}, \ldots, y_{t-n_a}, u_{t-1}, \ldots, u_{t-n_b})$

- **Neural state-space** models:

  - **w/ state data**: fit a neural network model $x_{t+1} \approx \mathcal{N}_x(x_t, u_t), \quad y_t \approx \mathcal{N}_y(x_t)$

  - **I/O data only**: set x_t = value of an inner layer of the network (Prasad, Bequette, 2003) such as an **autoencoder** (Masti, Bemporad, 2021)

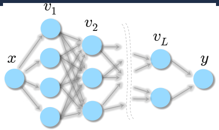
- **Recurrent neural networks** (RNNs): more appropriate for open-loop prediction, but more difficult to train than feedforward NNs



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

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

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



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

$$\theta = (A_1, b_1, \ldots, A_L, b_L)$$

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

$$
\begin{aligned}
\min_{\substack{\theta_x, \theta_y \\ x_0, x_1, \ldots, x_{N-1}}} \quad & r(x_0, \theta_x, \theta_y) + \frac{1}{N}\sum_{k=0}^{N-1} \ell(y_k, f_y(x_k, \theta_y)) \\
\text{s.t.} \quad & x_{k+1} = f_x(x_k, u_k, \theta_x)
\end{aligned}
$$

- **Main issue**: x_k are **hidden states**, i.e., are **unknowns** of the problem

TRAINING RNNS VIA EXTENDED KALMAN FILTERING

TRAINING RNNS BY EKF

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

- Estimate both hidden states x_k and parameters θ_x, θ_y by **EKF** based on model

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

Ratio $\mathrm{Var}[\eta_k]/\ \mathrm{Var}[\zeta_k]$ related to **learning-rate** of training algorithm

Inverse of initial matrix P_0 related to ℓ_2**-penalty** on θ_x, θ_y

- RNN and its hidden state x_k can be estimated **on line** from a streaming dataset $\{u_k, y_k\}$, and/or **offline** by processing multiple epochs of a given dataset

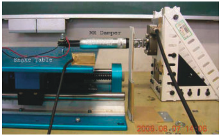
- Can handle **general smooth strongly convex** loss fncs/regularization terms

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

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

TRAINING RNNS BY EKF - EXAMPLES

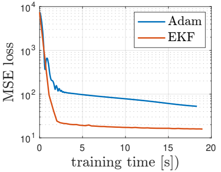
- **Dataset**: **magneto-rheological fluid damper**
  3499 I/O data (Wang, Sano, Chen, Huang, 2009)



- N=2000 data used for training, 1499 for testing the model

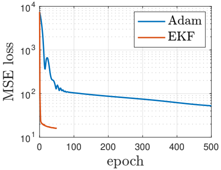
- Same data used in NNARX modeling demo of SYS-ID Toolbox for MATLAB

- **RNN model**: **4** hidden states, shallow
  state-update and output functions
  6 neurons, **atan** activation, I/O feedthrough



- Compare with gradient descent (Adam)

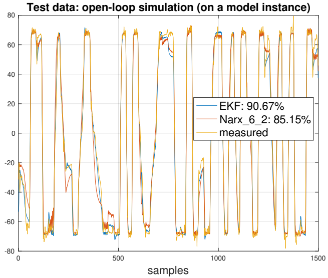
  MATLAB+CasADi implementation (Macbook Pro 14" M1 Max)



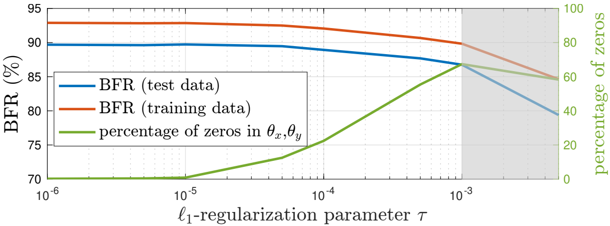
TRAINING RNNS BY EKF - EXAMPLES

- Compare BFR[1] 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 training with ℓ_1-penalty $\tau \left\| \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} \right\|_1$



[1]Best fit rate BFR=$100(1 - \frac{\|Y - \hat{Y}\|_2}{\|Y - \bar{y}\|_2})$, averaged over 20 runs from different initial weights

TRAINING RNNS VIA SEQUENTIAL LEAST SQUARES

TRAINING RNNS BY SEQUENTIAL LEAST-SQUARES

(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

- RNN training problem = **optimal control** problem:

$$\begin{array}{rl} \min_{\theta_x,\theta_y,x_0,x_1,\ldots,x_{N-1}} & r(x_0,\theta_x,\theta_y)+\sum_{k=0}^{N-1}\ell(y_k,\hat{y}_k) \\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x) \\ & \hat{y}_k=f_y(x_k,u_k,\theta_y) \end{array}$$

  - θ_x,θ_y,x_0 = manipulated variables, $\hat{y}_k$ = output, y_k = reference, u_k = meas. dist.
  - $r(x_0,\theta_x,\theta_y)$ = input penalty, $\ell(y_k,\hat{y}_k)$ = output penalty
  - N = prediction horizon, control horizon = 1

- **Linearized model**: given a current guess $\theta_x^h,\theta_y^h,x_0^h,\ldots,x_{N-1}^h$, approximate

$$\begin{array}{rcl} \Delta x_{k+1} & = & (\nabla_x f_x)'\Delta x_k+(\nabla_{\theta_x} f_x)'\Delta\theta_x \\ \Delta y_k & = & (\nabla_{x_k} f_y)'\Delta x_k+(\nabla_{\theta_y} f_y)'\Delta\theta_y \end{array}$$

TRAINING RNNS BY SEQUENTIAL LEAST-SQUARES

- Linearized dynamic response: $\Delta x_k = M_{kx}\Delta x_0 + M_{k\theta_x}\Delta\theta_x$

$$\begin{aligned} M_{0x} &= I, \quad M_{0\theta_x} = 0 \\ M_{(k+1)x} &= \nabla_x f_x(x_k^h, u_k, \theta_x^h) M_{kx} \\ M_{(k+1)\theta_x} &= \nabla_x f_x(x_k^h, u_k, \theta_x^h) M_{k\theta_x} + \nabla_{\theta_x} f_x(x_k^h, u_k, \theta_x^h) \end{aligned}$$

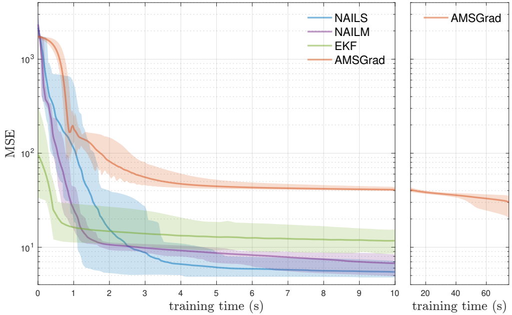
- Take $2^{\rm nd}$-order expansion of the loss ℓ and regularization term r
- Solve **least-squares** problem to get increments $\Delta x_0, \Delta\theta_x, \Delta\theta_y$
- Update $x_0^{h+1}, \theta_x^{h+1}, \theta_y^{h+1}$ by applying either a
  - **line-search** (LS) method based on Armijo rule
  - or a **trust-region** method (Levenberg-Marquardt) (LM)
- The resulting training method is a **Generalized Gauss-Newton** method

$\Rightarrow$ very good convergence properties (Messerer, Baumgärtner, Diehl, 2021)

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

- Fluid-damper example: (4 states, shallow NNs w/ **4 neurons**, **I/O feedthrough**)



MSE loss on training data, mean value and range over 20 runs from different random initial weights

NAILS = GNN method with line search

NAILM = GNN method with LM steps

BFR	training	test
NAILS	94.41 (0.27)	89.35 (2.63)
NAILM	94.07 (0.38)	89.64 (2.30)
EKF	91.41 (0.70)	87.17 (3.06)
AMSGrad	84.69 (0.15)	80.56 (0.18)

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

- We also want to handle **non-smooth** (and **non-convex**) regularization terms

$$\begin{array}{rl}\min_{\theta_x,\theta_y,x_0} & r(x_0,\theta_x,\theta_y)+\sum_{k=0}^{N-1}\ell(y_k,f_y(x_k,\theta_y))+g(\theta_x,\theta_y)\\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x)\end{array}$$

- **Idea**: use **alternating direction method of multipliers** (ADMM) by splitting

$$\begin{array}{rl}\min_{\theta_x,\theta_y,x_0,\nu_x,\nu_y} & r(x_0,\theta_x,\theta_y)+\sum_{k=0}^{N-1}\ell(y_k,f_y(x_k,\theta_y))+g(\nu_x,\nu_y)\\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x)\\ & \begin{bmatrix}\nu_x\\ \nu_y\end{bmatrix}=\begin{bmatrix}\theta_x\\ \theta_y\end{bmatrix}\end{array}$$

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

(Bemporad, 2021 - http://arxiv.org/abs/2112.15348)

- ADMM + Seq. LS = **NAILS** algorithm (Nonconvex ADMM Iterations and Sequential LS)

$$\begin{bmatrix} x_0^{t+1} \\ \theta_x^{t+1} \\ \theta_y^{t+1} \end{bmatrix} = \arg\min_{x_0,\theta_x,\theta_y} V(x_0,\theta_x,\theta_y) + \frac{\rho}{2}\left\|\begin{bmatrix} \theta_x-\nu_x^t+w_x^t \\ \theta_y-\nu_y^t+w_y^t \end{bmatrix}\right\|_2^2$$

(sequential) LS

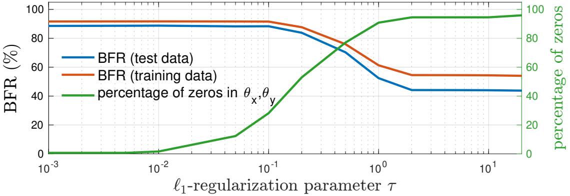
$$\begin{bmatrix} \nu_x^{t+1} \\ \nu_y^{t+1} \end{bmatrix} = \mathrm{prox}_{\frac{1}{\rho}g}(\theta_x^{t+1}+w_x^t, \theta_y^{t+1}+w_y^t)$$

proximal step

$$\begin{bmatrix} w_x^{t+1} \\ w_y^{t+1} \end{bmatrix} = \begin{bmatrix} w_x^h+\theta_x^{t+1}-\nu_x^{t+1} \\ w_y^h+\theta_y^{t+1}-\nu_y^{t+1} \end{bmatrix}$$

update dual vars

- Fluid-damper example: **Lasso regularization** $g(\nu_x,\nu_y) = \tau_x\|\nu_x\|_1 + \tau_y\|\nu_y\|_1$



$\tau_x = \tau_y = \tau$

(mean results over 20 runs from different initial weights)

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

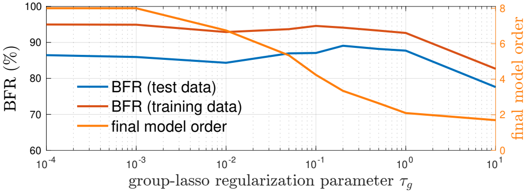
(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

- Fluid-damper example: **Lasso regularization** $g(\nu_x, \nu_y) = 0.2\|\nu_x\|_1 + 0.2\|\nu_y\|_1$

training algorithm	BFR training	BFR test	sparsity %	CPU time	# epochs
NAILS	91.00 (1.66)	87.71 (2.67)	**65.1** (6.5)	**11.4** s	250
NAILM	**91.32** (1.19)	**87.80** (1.86)	64.1 (7.4)	11.7 s	250
EKF	89.27 (1.48)	86.67 (2.71)	47.9 (9.1)	13.2 s	50
AMSGrad	91.04 (0.47)	88.32 (0.80)	16.8 (7.1)	64.0 s	2000
Adam	90.47 (0.34)	87.79 (0.44)	8.3 (3.5)	63.9 s	2000
DiffGrad	90.05 (0.64)	87.34 (1.14)	7.4 (4.5)	63.9 s	2000

$\approx$ same fit than SGD/EKF but sparser models and faster (CPU: Apple M1 Pro)

- Fluid-damper example: **group-Lasso regularization** $g(\nu_i^g) = \tau_g \sum_{i=1}^{n_x} \|\nu_i^g\|_2$ to zero entire rows and columns and **reduce state-dimension** automatically

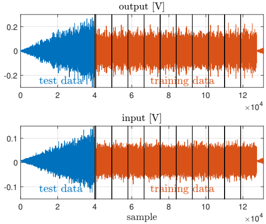


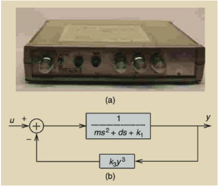
good choice: $n_x = 3$
(best fit on test data)

TRAINING RNNS - SILVERBOX BENCHMARK

(Wigren, Schoukens, 2013)

- **Silverbox benchmark** (Duffin oscillator): 10 traces of ≈8600 data used for training, 40000 for testing (`http://www.nonlinearbenchmark.org`)





(Schoukens, Ljung, 2019)

- **RNN model**: 8 states, 3 layers of 8 neurons, atan activation, no I/O feedthrough

- **Initial-state encoder**: NN with 2 layers of 4 neurons, fed by 8 past inputs + 8 past outputs, atan activation (Beintema, Toth, Schoukens, 2021) (Masti, Bemporad, 2021)

- Total number of parameters $n_{\theta_x} + n_{\theta_y} + n_{\theta_{x0}}$ =296+225+128=649

TRAINING RNNS - SILVERBOX BENCHMARK

(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

- Identification results on test data [2]:

identification method	RMSE [mV]	BFR [%]
ARX (ml) [1]	16.29 [4.40]	69.22 [73.79]
NLARX (ms) [1]	8.42 [4.20]	83.67 [92.06]
NLARX (mlc) [1]	1.75 [1.70]	96.67 [96.79]
NLARX (ms8c50) [1]	1.05 [0.30]	98.01 [99.43]
Recurrent LSTM model [2]	2.20	95.83
SS encoder [3] ($n_x = 4$)	[1.40]	[97.35]
NAILM	0.35	99.33

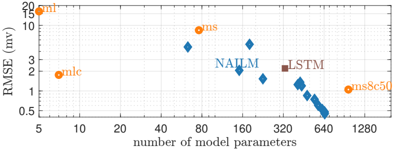
[1] Ljung, Zhang, Lindskog, Juditski, 2004

[2] Ljung, Andersson, Tiels, Schön, 2020

[3] Beintema, Toth, Schoukens, 2021

- NAILM training time ≈ 400 s (MATLAB+CasADi on Apple M1 Max CPU)

- Repeat training with ℓ_1-regularization:



[2] Trained RNN: `http://cse.lab.imtlucca.it/~bemporad/shared/silverbox/rnn888.zip`

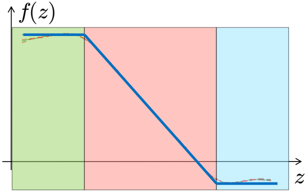
PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

PWA REGRESSION PROBLEM

- **Problem**: Given input/output pairs $\{x(k), y(k)\}$, $k = 1, \ldots, N$ and number s of models, compute a **piecewise affine** (PWA) approximation $y \approx f(x)$

$$v(k) = \begin{cases} F_1 z(k) + g_1 & \text{if } H_1 z(k) \leq K_1 \\ \vdots & \\ F_s z(k) + g_s & \text{if } H_s z(k) \leq K_s \end{cases}$$

$$v(k) = \begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix}, \quad z(k) = \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}$$



- Quite rich literature on PWA identification (Breiman, 1993) (Münz, Krebs, 2002) (Ferrari-Trecate, Muselli, Liberati, Morari, 2003) (Juloski, Wieland, Heemels, 2004) (Roll, Bemporad, Ljung, 2004) (Bemporad, Garulli, Paoletti, Vicino, 2005) (Pillonetto, 2016) (Breschi, Piga, Bemporad, 2016)

- Any **ML technique** can be applied that leads to PWA models, such as **(leaky-)ReLU-NNs**, **decision trees**, **softmax regression**, **KNN**, ...

PARC - PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

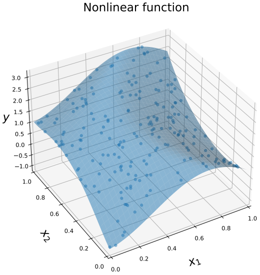
(Bemporad, 2022)

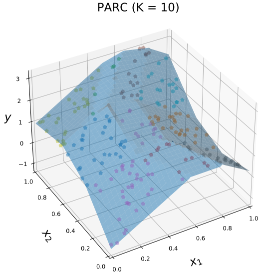
- New **Piecewise Affine Regression and Classification** (**PARC**) algorithm
- Training dataset:
  - **feature vector** $z \in \mathbb{R}^n$ (categorical features **one-hot encoded** in $\{0, 1\}$)
  - **target vector** $v_c \in \mathbb{R}^{m_c}$ (numeric), $v_{di} \in \{w_{di}^1, \ldots, w_{di}^{m_i}\}$ (categorical)
- PARC iteratively **clusters** training data in K sets and **fits** linear predictors:
  1. fit $v_c = a_j z + b_j$ by **ridge regression** (=ℓ_2-regularized least squares)
  2. fit $v_{di} = w_{di}^{h_*}, h_* = \arg\max\{a_{dih}^h z + b_{di}^h\}$ by **softmax regression**
  3. fit a convex **PWL separation function** by **softmax regression**

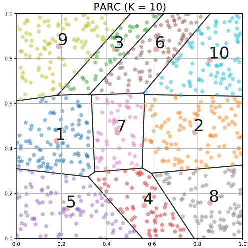
$$\Phi(z) = \omega^{j(z)} z + \gamma^{j(z)}, \qquad j(z) = \min\left\{\arg\max_{j=1,\ldots,K}\{\omega^j z + \gamma^j\}\right\}$$

- Data reassigned to clusters based on weighted fit/PWL separation criterion
- PARC is a **block-coordinate descent** algorithm $\Rightarrow$ (local) convergence ensured

- Simple PWA regression example:
  - 1000 samples of $y = \sin(4x_1 - 5(x_2 - 0.5)^2) + 2x_2$ (use 80% for training)
  - Look for PWA approximation over $K = 10$ polyhedral regions

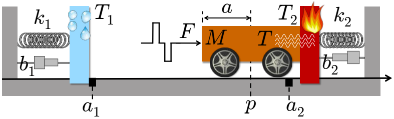






- Code download: **http://cse.lab.imtlucca.it/~bemporad/parc/**

PARC - CART & BUMPERS EXAMPLE



- **Example**: moving cart and bumpers + heat transfer during bumps.

  Spring and viscous forces are **nonlinear**.

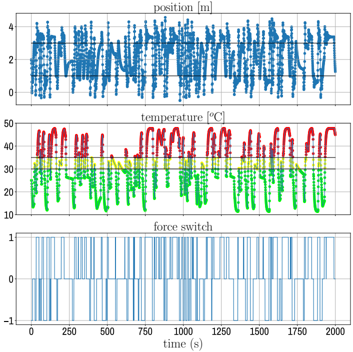
- Categorical input $F \in \{-\bar{F}, 0, \bar{F}\}$ and categorical output $c \in \{green, yellow, red\}$

- Continuous-time system simulated for 2,000 s, sample time = 0.5 s (=4000 training samples)

- Feature vector $z_k = [y_k, \dot{y}_k, T_k, F_k]$

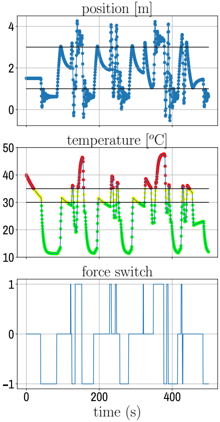
- Target vector $v_k = [y_{k+1}, \dot{y}_{k+1}, T_{k+1}, c_k]$

- Hybrid model learned by **PARC** ($K = 5$ regions)

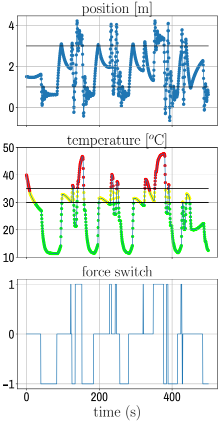


PARC - CART & BUMPERS EXAMPLE

- **Open-loop** simulation on 500 s **test** data:



continuous-time system

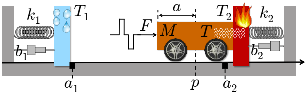


discrete-time PWA model

- Model fit is good enough for MPC design purposes (see next slide ...)

PARC - CART & BUMPERS EXAMPLE

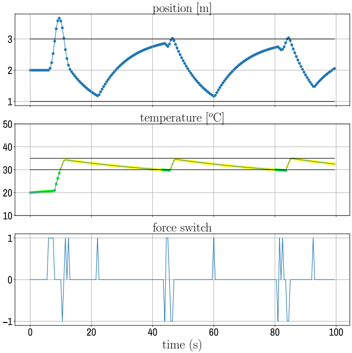
- MPC problem with prediction horizon $N = 9$:



$$\min_{F_0,\ldots,F_{N-1}} \quad \sum_{k=0}^{N-1} |c_k - 1| + 0.25|F_k|$$
$$\text{s.t.} \quad F_k \in \{-\bar{F}, 0, \bar{F}\}$$
PWA model equations

- MILP solution time: 0.37-1.9 s (CPLEX)

- **Data-driven hybrid MPC** controller can keep temperature in yellow zone



- **Approximate explicit MPC**: fit a **decision tree** on 10,000 samples (accuracy: 99.7%). CPU time = 73÷88 μs. Closed-loop trajectories very similar.

LEARNING OPTIMAL MPC CALIBRATION

MPC CALIBRATION PROBLEM

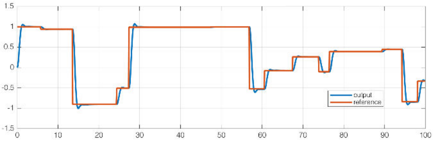
- The design depends on a vector x of **MPC parameters**

- 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)



- **Auto-tuning** = find the best combination of parameters by solving the **global optimization problem**

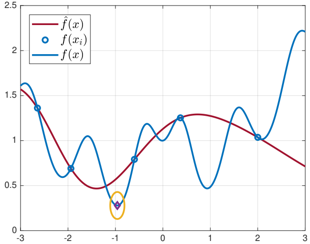
$$\min_x f(x)$$

- 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, 2020)

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

```
pip install 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 $x_1, \ldots, x_{N_{\text{init}}}$ **(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: $\phi(\epsilon d) = \frac{1}{1+(\epsilon d)^2}$

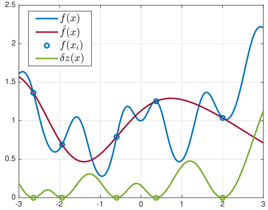
(inverse quadratic)

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!

AUTO-TUNING - GLIS

- **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**

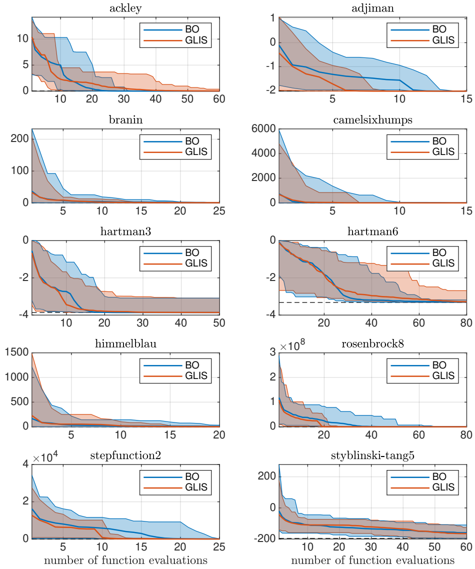
$$\begin{array}{rcl} x_{N+1} = & \arg\min & \hat{f}(x) - \delta z(x) \\ & \text{s.t.} & \ell \leq x \leq u,\ g(x) \leq 0 \end{array}$$

δ= *exploitation* vs *exploration* tradeoff

to get new sample x_{N+1}

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

GLIS VS BAYESIAN OPTIMIZATION



problem	n	BO [s]	GLIS [s]
ackley	2	29.39	3.13
adjiman	2	3.29	0.68
branin	2	9.66	1.17
camelsixhumps	2	4.82	0.62
hartman3	3	26.27	3.35
hartman6	6	54.37	8.80
himmelblau	2	7.40	0.90
rosenbrock8	8	63.09	13.73
stepfunction2	4	11.72	1.81
styblinski-tang5	5	37.02	6.10

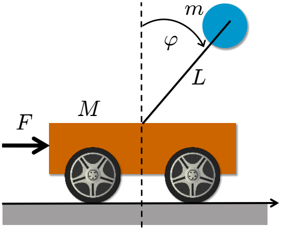
Results computed on 20 runs per test

BO = MATLAB's **bayesopt** fcn

MPC AUTOTUNING EXAMPLE

(Forgione, Piga, Bemporad, 2020)

- Linear MPC applied to cart-pole system: **14 parameters** to tune

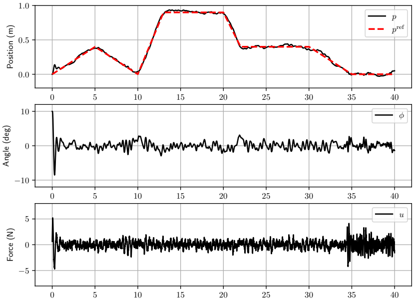


  - **sample time**
  - **weights** on outputs and input increments
  - prediction and control **horizons**
  - **covariance** matrices of Kalman filter
  - absolute and relative **tolerances** of QP solver

- Closed-loop performance score: $J = \int_0^T |p(t) - p_{\rm ref}(t)| + 30|\phi(t)|dt$
- MPC parameters tuned using 500 iterations of GLIS
- Performance tested with simulated cart on two hardware platforms (PC, Raspberry PI)

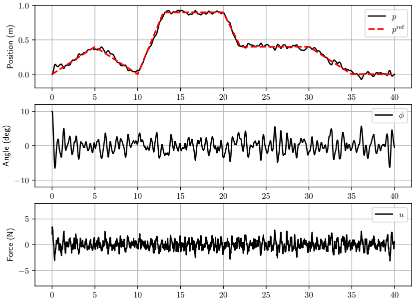
MPC AUTOTUNING EXAMPLE

MPC optimized for **desktop PC**



optimal sample time = **6 ms**

MPC optimized for **Raspberry PI**



optimal sample time = **22 ms**

- MPC parameters tuned by **GLIS** global optimizer (500 fcn evals)
- Auto-calibration can squeeze max performance out of the available hardware
- Bayesian optimization gives similar results, but with larger computation effort

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, ...)
  - 👍 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 have **multiple objectives**, not clear how to blend them in a single one

ACTIVE PREFERENCE LEARNING

(Bemporad, Piga, *Machine Learning*, 2021)

- Objective function $f(x)$ is not available (**latent function**)
- We can only express a **preference** between two choices:

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

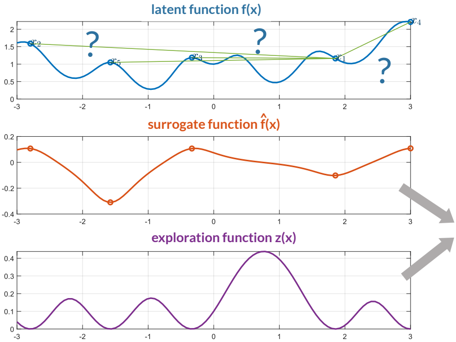
- We want to find a global optimum $x^\star$ (="better" than any other x)

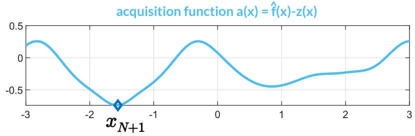
$$\text{find } x^\star \text{ such that } \pi(x^\star, x) \leq 0,\ \forall x \in \mathcal{X},\ \ell \leq x \leq u$$

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

ACTIVE PREFERENCE LEARNING ALGORITHM

(Bemporad, Piga, *Machine Learning*, 2021)

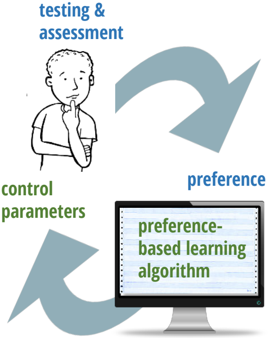




- **Fit a surrogate** $\hat{f}(x)$ that respects the **preferences** expressed by the decision maker at sampled points (by solving a QP)
- **Minimize an acquisition function** $\hat{f}(x) - \delta z(x)$ to get a **new sample** x_{N+1}
- **Compare** x_{N+1} to the current "best" point and **iterate**

SEMI-AUTOMATIC CALIBRATION BY PREFERENCE-BASED LEARNING

- Use **preference-based optimization** (**GLISp**) algorithm for **semi-automatic tuning** of MPC (Zhu, Bemporad, Piga, 2021)
- Latent function = calibrator's (unconscious) score of closed-loop MPC performance
- GLISp **proposes a new combination** x_{N+1} of MPC parameters to test
- By observing test results, the calibrator expresses a **preference**, telling if x_{N+1} is "**better**", "**similar**", or "**worse**" than current best combination
- Preference learning algorithm: **update the surrogate** $\hat{f}(x)$ of the latent function, optimize the acquisition function, **ask preference**, and **iterate**

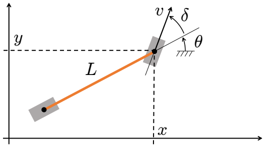


PREFERENCE-BASED TUNING: MPC EXAMPLE

(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:

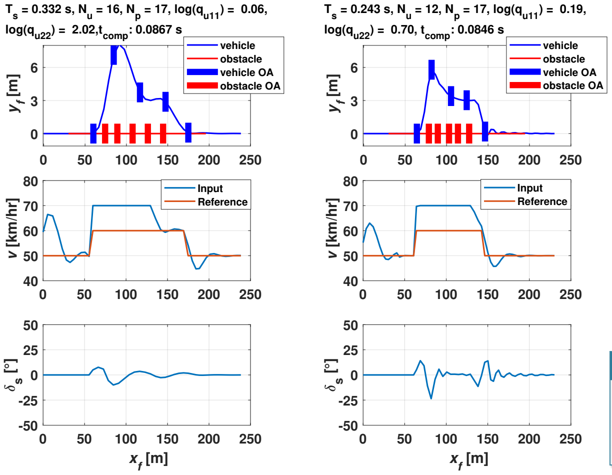
"*optimal obstacle avoidance*", "*pleasant drive*", "*CPU time small enough*", …

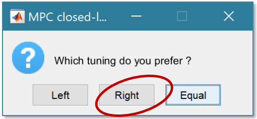
not easy to quantify in a single function

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

PREFERENCE-BASED TUNING: MPC EXAMPLE

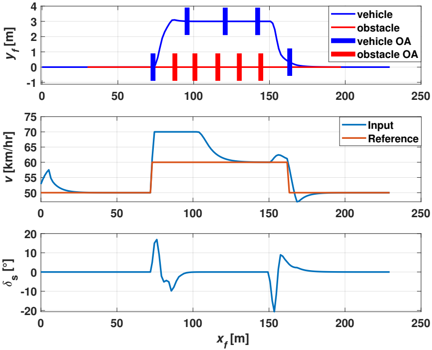
- Preference query window:





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!

- Extended to handle also **unknown constraints** (Zhu, Piga, Bemporad, 2021)

CORNER-CASE DETECTION

CORNER-CASE DETECTION PROBLEM

(Zhu, Bemporad, Kneissl, Esen, 2022)

- **Goal**: detect undesired simulation scenarios (=corner-cases)
- Let x = parameters defining the scenario, $\mathcal{X}_{\rm ODD}$ = operational design domain
  $x \in \mathcal{X}_{\rm ODD} \subseteq \mathbb{R}^n$
- critical scenario = vector x^* for which the closed-loop behavior is critical
- Example:
  - x = (initial distance between ego car and obstacle, obstacle acceleration, ...)
  - Critical scenario: time-to-collision is too short, excessive jerk of ego car, ...
- **Key idea**: use global optimizer GLIS to generate critical corner-cases

$$\begin{array}{rl} x^* \in \arg\min\limits_{x\in\mathcal{X}_{\rm ODD}} & f(x) \\ \text{s.t.} & \ell \leq x \leq u \end{array}$$

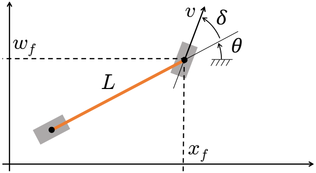
$f(x)$ = criticality of closed-loop simulation (or experiment) determined by scenario x (the smaller $f(x)$, the more critical x is)

CORNER-CASE DETECTION: CASE STUDY

- **Problem**: find critical scenarios in automated driving w/ obstacles
- **MPC controller** for lane-keeping and obstacle-avoidance based on simple kinematic bicycle model (Zhu, Piga, Bemporad, 2021)

$$\begin{aligned}\dot{x}_f &= v\cos(\theta+\delta)\\ \dot{w}_f &= v\sin(\theta+\delta)\\ \dot{\theta} &= \frac{v\sin(\delta)}{L}\end{aligned}$$

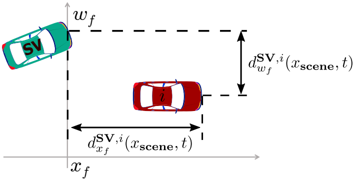
(x_f, w_f) = front-wheel position



- **Black-box optimization** problem: given k obstacles, solve

$$\min_{\ell\le x\le u} \quad \sum_{i=1}^{k} d^{\text{SV},i}_{x_f,\text{critical}}(x) + d^{\text{SV},i}_{w_f,\text{critical}}(x)$$

s.t. other constraints



CORNER-CASE DETECTION: CASE STUDY

- **Cost function terms** to minimize: for each obstacle $\#i$ define

$$d^{\mathrm{SV},i}_{x_f,\mathrm{critical}}(x) = \begin{cases} \min\limits_{t\in T_{\mathrm{collision}}} d^{\mathrm{SV},i}_{x_f}(x,t) & \mathcal{I}^i_{\mathrm{collision}} \\ L & \sim\mathcal{I}^i_{\mathrm{collision}}\ \&\ \mathcal{I}_{\mathrm{collision}} \\ \sum\limits_{t\in T_{\mathrm{sim}}} d^{\mathrm{SV},i}_{x_f}(x,t) & \sim\mathcal{I}_{\mathrm{collision}} \end{cases}$$

min time of collision with $\#i$

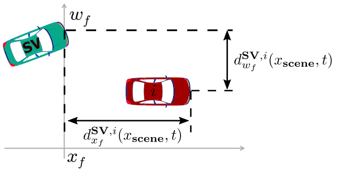
collision with other $\#j \neq \#i$

no collision

$$d^{\mathrm{SV},i}_{w_f,\mathrm{critical}}(x) = \begin{cases} \min\limits_{t\in T_{\mathrm{collision}}} d^{\mathrm{SV},i}_{w_f}(x,t) & \mathcal{I}^i_{\mathrm{collision}} \\ w_{f,\mathrm{safe}} & \sim\mathcal{I}^i_{\mathrm{collision}}\ \&\ \mathcal{I}_{\mathrm{collision}} \\ \sum\limits_{t\in T_{\mathrm{sim}}} d^{\mathrm{SV},i}_{w_f}(x,t) & \sim\mathcal{I}_{\mathrm{collision}} \end{cases}$$

$$\mathcal{I}^i_{\mathrm{collision}} = \texttt{true} \quad \text{if} \quad \exists t \in T_{\mathrm{sim}} \text{ s.t.}$$
$$(d^{\mathrm{SV},i}_{x_f}(x,t) \leq L)\ \&\ (d^{\mathrm{SV},i}_{w_f}(x,t) \leq W)$$

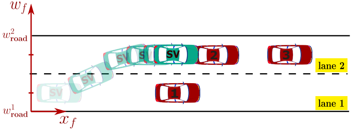
$$\mathcal{I}_{\mathrm{collision}} = \texttt{true} \quad \text{if} \quad \exists h \quad \text{s.t.} \quad \mathcal{I}^h_{\mathrm{collision}} = \texttt{true}$$



- **Logical scenario 1**: GLIS identifies 64 collision cases within 100 simulations

iter	x					
	x_{f1}^0	v_1^0	x_{f2}^0	v_2^0	x_{f3}^0	v_3^0
51	15.00	30.00	44.14	10.00	49.10	47.39
79	28.09	30.00	70.29	10.00	74.79	31.74
40	34.30	30.00	60.59	10.00	77.80	35.97

red = optimal solution found by GLIS solver

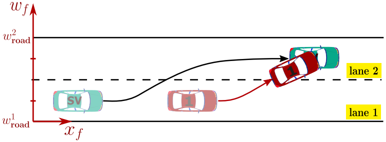


Ego car changes lane to avoid #1, but cannot brake fast enough to avoid #2

- **Logical scenario 2**: GLIS identifies 9 collision cases within 100 simulations

iter	x		
	x_{f1}^0	v_1^0	t_c
28	12.57	46.94	16.75
16	17.53	47.48	23.65
88	44.54	41.26	16.02

red = optimal solution found by GLIS solver



Ego car changes lane to avoid #1, but cannot decelerate in time for the sudden lane-change of #1

ACTIVE LEARNING

ACTIVE LEARNING ALGORITHMS

- How to **select the training samples** to train a good model?
  (problem related to **design of experiment** (Fisher, 1935))

- **Active learning** (AL) algorithms select the feature vectors x_k to query for the corresponding target y_k while training based on the model learned so far (Settles, 2012)

- New AL algorithm: **IDEAL** (**I**nverse-**D**istance based **E**xploration for **A**ctive **L**earning) (Bemporad, 2023)

- Code download: **http://cse.lab.imtlucca.it/~bemporad/ideal/**

ACTIVE-LEARNING METHOD "IDEAL" FOR REGRESSION

(Bemporad, 2023)

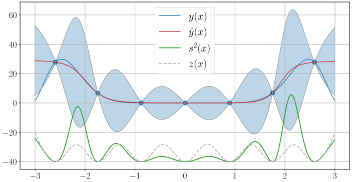
- First generate N random samples $\{x_k\}$ and acquire corresponding $\{y_k\}$
- Fit model $\hat{y}(x)$ based on $(x_1, y_1), \ldots, (x_N, y_N)$
- **Similar to GLIS**, acquire new sample by maximizing the **acquisition function**

$$x_{N+1} = \arg\max_{x \in \mathcal{X}_P} s^2(x) + \delta z(x) \qquad s^2(x) = \sum_{k=1}^{N} v_k(x)(y_k - \hat{y}(x))^2$$

exploration/exploitation tradeoff

IDW variance function (Joseph, Kang, 2011)

- Fit new model $\hat{y}(x)$ based on $(x_1, y_1), \ldots$ (x_{N+1}, y_{N+1})
- Iterate, until max # querable samples reached

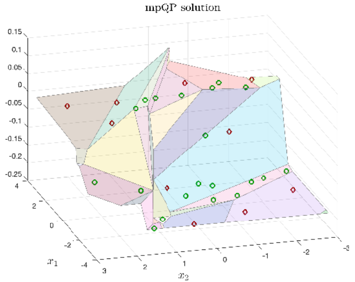


ACTIVE LEARNING EXAMPLE: EXPLICIT MPC

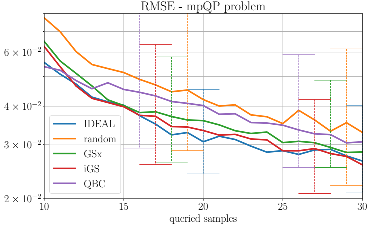
- We want to approximate the solution of the **multiparametric QP problem**

$$
\begin{array}{rcllll}
z^*(x) & = & \arg\min_z & \frac{1}{2}z'Qz + x'F'z & x \in \mathbb{R}^2 & -3 \le x_i \le 3 \\
 & & \text{s.t.} & Az \le b + Sx & b \in \mathbb{R}^{12} & \\
 & & & \ell \le z \le u & z \in \mathbb{R}^{12} & \\
y(x) & = & [1\,0\,\ldots\,0]\,z^*(x) & & &
\end{array}
$$

- **Goal**: Actively learn $\hat{y}(x)$ = NN with ReLU activation and (10,10,10) neurons



○ = queried samples, ◇ = initial random samples



GS_x = (Yu, Kim, 2010)

iGS = (Wu, Lin, Huang, 2019)

QBC = (Burbidge, Rowland, King, 2007)

CONCLUSIONS

CONCLUSIONS

- **ML** very useful to get **control-oriented models** (and **control laws**) from **data**

- **ML** cannot replace control engineering:
  - Blindly applying deep NNs can lead to useless models for embedded control
  - Approximating MPC laws by NN's can fail, often still need online optimization
  - Model-free **reinforcement learning** can fail wrt model-based control design

- Ignoring **ML** tools would be a mistake (a lot to "learn" from machine learning)

- A wide spectrum of research opportunities and new practices is open !