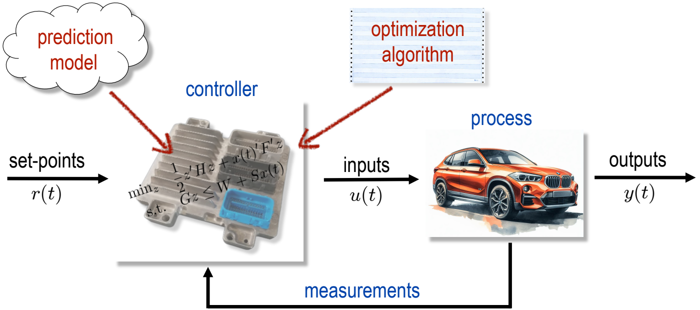
LEARNING-BASED METHODS FOR MODEL PREDICTIVE CONTROL

Alberto Bemporad

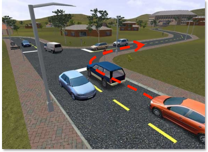
imt.lu/ab

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

October 12, 2023



- **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

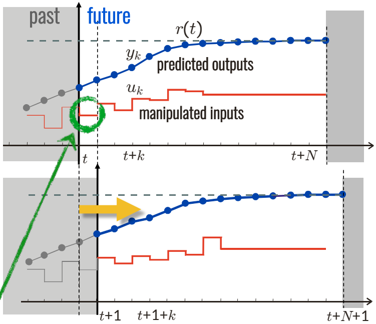


- **MPC problem**: find the best control sequence over a future horizon of N steps

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

$$\begin{array}{lll} \text{s.t.} & x_{k+1} = f(x_k, u_k) & \text{prediction model} \\ & y_k = g(x_k) & \\ & u_{\min} \le u_k \le u_{\max} & \text{constraints} \\ & y_{\min} \le y_k \le y_{\max} & \\ & x_0 = x(t) & \text{state feedback} \end{array}$$

→ **numerical optimization problem**

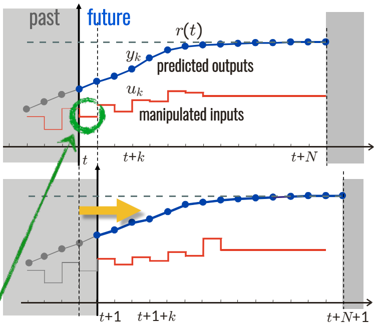


MODEL PREDICTIVE CONTROL

- **MPC problem**: find the best control sequence over a future horizon of N steps

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

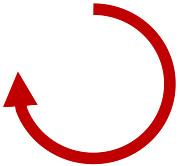
$$\begin{aligned} \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) && \text{prediction model} \\ & y_k = g(x_k) \\ & u_{\min} \le u_k \le u_{\max} && \text{constraints} \\ & y_{\min} \le y_k \le y_{\max} \\ & x_0 = x(t) && \text{state feedback} \end{aligned}$$



numerical optimization problem

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

Repeat at all time steps t

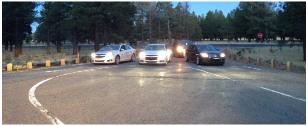


MPC IN INDUSTRY

- Conceived in the 60's (Rafal, Stevens, 1968) (Propoi, 1963)
- Used in the **process industries** since the 80's (Qin, Badgewell, 2003)
- Now massively spreading to the automotive industry and other sectors

MPC IN INDUSTRY

- Conceived in the 60's (Rafal, Stevens, 1968) (Propoi, 1963)
- Used in the **process industries** since the 80's (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)

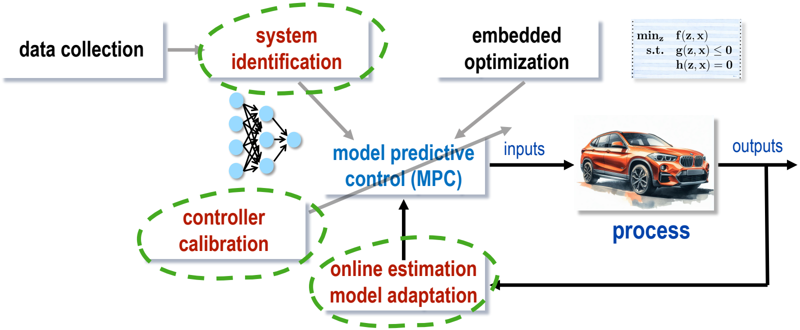


First known mass production of MPC in the automotive industry

ODYS
Advanced Controls & Optimization

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

RESEARCH ISSUES IN EMBEDDED MPC DESIGN



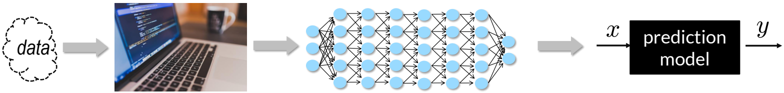
Focus of my talk:

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

LEARNING PREDICTION MODELS FOR MPC

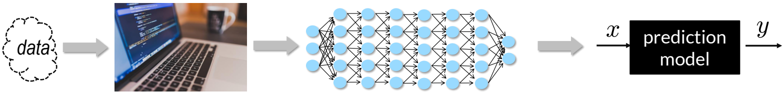
CONTROL-ORIENTED NONLINEAR MODELS

- **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**)



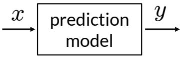
CONTROL-ORIENTED NONLINEAR MODELS

- **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**)



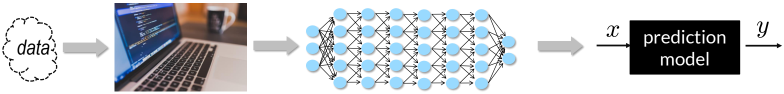
- **Physics-based** models: use physical principles to create a prediction model (**fewer parameters to learn, better generalizes on unseen data**)

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



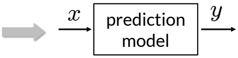
CONTROL-ORIENTED NONLINEAR MODELS

- **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**)

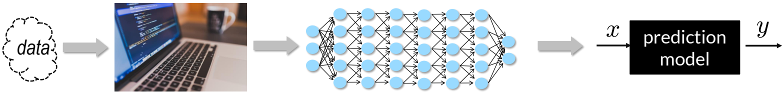
$$
\begin{aligned}
\dot{p}_1 &= k_1(W_c + W_{egr} - k_e p_1) + \frac{\dot{T}_1}{T_1} p_1 \\
\dot{p}_2 &= k_2(k_e p_1 - W_{egr} - W_t + W_f) + \frac{\dot{T}_2}{T_2} p_2 \\
\dot{P}_c &= \tfrac{1}{\tau}(P_c - \eta_m P_t)
\end{aligned}
$$



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

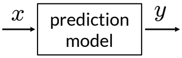
CONTROL-ORIENTED NONLINEAR MODELS

- **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**)

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



- **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 $y_t \approx \mathcal{N}(y_{t-1}, \ldots, y_{t-n_a}, u_{t-1}, \ldots, u_{t-n_b})$

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 $x_{t+1} \approx \mathcal{N}_x(x_t, u_t), \quad y_t \approx \mathcal{N}_y(x_t)$

NONLINEAR SYS-ID BASED ON NEURAL NETWORKS

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

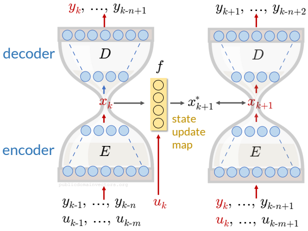
  (Narendra, Parthasarathy, 1990) (Hunt et al., 1992) (Suykens, Vandewalle, De Moor, 1996)

- **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)



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)

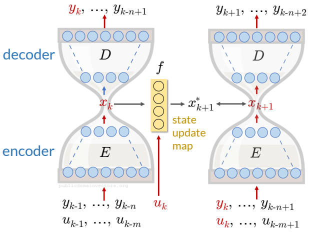
- **NNARX** models: use a **feedforward neural network** to approximate the nonlinear difference equation $y_t \approx \mathcal{N}(y_{t-1}, \ldots, y_{t-n_a}, u_{t-1}, \ldots, u_{t-n_b})$

- **Neural state-space** models:

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

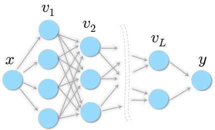
  - **I/O data only**: set x_t = value of an inner layer of the network (Prasad, Bequette, 2003) such as an **autoencoder** (Masti, Bemporad, 2021)

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



RECURRENT NEURAL NETWORKS

- **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, ...)

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

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

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

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

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

TRAINING RNNS 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

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

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

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

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

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

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 SEQUENTIAL LEAST-SQUARES

(Bemporad, 2023)

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

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

  - 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 $\Delta x_0, \Delta\theta_x, \Delta\theta_y$

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

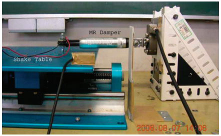
$\Rightarrow$ very good convergence properties (Messerer, Baumgärtner, Diehl, 2021)

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

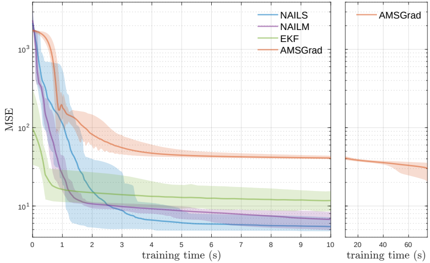
- **Example**: **magneto-rheological fluid damper**

  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 and range over 20 runs from different random initial weights

NAILS = GNN method with line search

NAILM = GNN method with LM steps

Best Fit Rate	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, 2023)

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

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

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

(Bemporad, 2023)

- 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 the alternating direction method of multipliers (ADMM)

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

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

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

NAILS - **N**onconvex **A**DMM **I**terations and sequential LS w/ **L**ine-**S**earch

NAILM - **N**onconvex **A**DMM **I**terations and sequential LS w/ **L**evenberg-**M**arquardt

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

(Bemporad, 2023)

- 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)

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

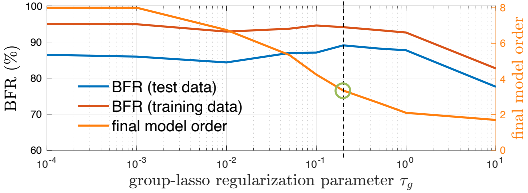
(Bemporad, 2023)

- 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

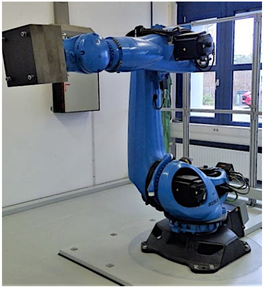


good choice: $n_x = 3$ (best fit on test data)

INDUSTRIAL ROBOT BENCHMARK

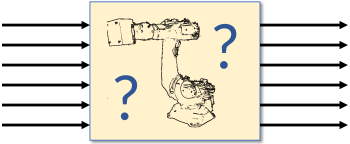
(Weigand, Götz, Ulmen, Ruskowski, 2022)

- KUKA KR300 R2500 ultra SE industrial robot, full robot movement
- **6 inputs** (torques), **6 outputs** (joint angles), backlash
- Identification benchmark dataset (forward model):
  - Sample time: $T_s = 100$ ms
  - N = 39988 training samples
  - N_{test} = 3636 test samples



`nonlinearbenchmark.org`

INDUSTRIAL ROBOT BENCHMARK: CHALLENGES



- Highly **nonlinear** dynamics.
  Nonlinear modeling required

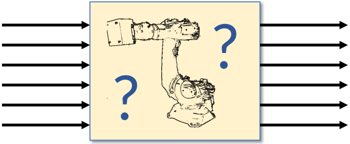
- **Multi-input / multi-output**, highly coupled system

- Data are **slightly over-sampled**, $\|y_k - y_{k-1}\|$ is often very small,
  need to minimize open-loop simulation error

- **Limited information**: easy to overfit training data and get poor testing results

- **Large number of samples** complicates numerical optimization

INDUSTRIAL ROBOT BENCHMARK: CHALLENGES



- Highly **nonlinear** dynamics.
  Nonlinear modeling required

- **Multi-input / multi-output**, highly coupled system

- Data are **slightly over-sampled**, $\|y_k - y_{k-1}\|$ is often very small,
  need to minimize open-loop simulation error

- **Limited information**: easy to overfit training data and get poor testing results

- **Large number of samples** complicates numerical optimization

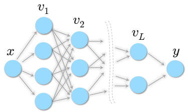
Finding a model that minimizes the simulation error is a rather challenging task from a computational viewpoint

RECURRENT NEURAL NETWORKS IN RESIDUAL FORM

(Bemporad, 2023 - *NLSYS-ID Benchmarks Workshop*)

- **Recurrent Neural Network** (RNN) model in **residual form**:

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



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

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

- **Goal**: minimize **open-loop simulation error** under **elastic net** regularization

$$\begin{aligned} \min_{A,B,C,\theta_x,\theta_y} \ & \frac{1}{N}\sum_{k=1}^{N} \|y_k - \hat{y}_k\|_2^2 + \frac{1}{2}\rho(\|\theta_x\|_2^2 + \|\theta_y\|_2^2) + \tau(\|\theta_x\|_1 + \|\theta_y\|_1) \\ \text{s.t.} \ & \text{model equations}, \ x_0 = 0 \end{aligned}$$

- **ℓ_1-regularization** introduced to reduce # model coefficients (=simpler model)

SOLUTION APPROACH

(Bemporad, 2023 - *NLSYS-ID Benchmarks Workshop*)

1. Standard-scale I/O data for numerical reasons $u_i \leftarrow \frac{u_i - \mu_u^i}{\sigma_u^i}$, $y_i \leftarrow \frac{y_i - \mu_y^i}{\sigma_y^i}$ $i = 1, \ldots, 6$

2. Train (A, B, C) by N4SID (Overschee, De Moor, 1994) with focus on simulation

3. Train simple RESNET model with shallow NNs:

$$x_{k+1} = Ax_k + Bu_k + f_x(x_k, u_k, \theta_x), \qquad y_k = Cx_k + f_y(x_k, \theta_y)$$

- **Optimization setup**: in Python, using **JAX** and **L-BFGS-B** (Byrd, Lu, Nocedal, Zhu, 1995) to handle ℓ_1-regularization

TRAINING RNN W/ ℓ_1-PENALTIES VIA L-BFGS-B

- To handle ℓ_1-regularization, split $\theta_x = \theta_x^+ - \theta_x^-$ and $\theta_y = \theta_y^+ - \theta_y^-$:

$$\min_{\theta_x^+,\theta_y^+,\theta_x^-,\theta_y^-} \frac{1}{N}\sum_{k=1}^{N} \|y_k - \hat{y}_k\|_2^2 + \frac{1}{2}\rho \left\| \begin{bmatrix} \theta_x^+ \\ \theta_y^+ \\ \theta_x^- \\ \theta_y^- \end{bmatrix} \right\|_2^2 + \tau \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}' \begin{bmatrix} \theta_x^+ \\ \theta_y^+ \\ \theta_x^- \\ \theta_y^- \end{bmatrix}$$

$$\text{s.t. model equations}, \ x_0 = 0$$

$$\theta_x^+, \theta_y^+, \theta_x^-, \theta_y^- \geq 0$$

- **Lemma**: Weighting $\|\theta_x^+\|_2^2 + \|\theta_x^-\|_2^2 + \|\theta_y^+\|_2^2 + \|\theta_y^-\|_2^2$ is equivalent to weighting $\|\theta_x^+ - \theta_x^-\|_2^2 + \|\theta_y^+ - \theta_y^-\|_2^2$ (proof is simple by contradiction)

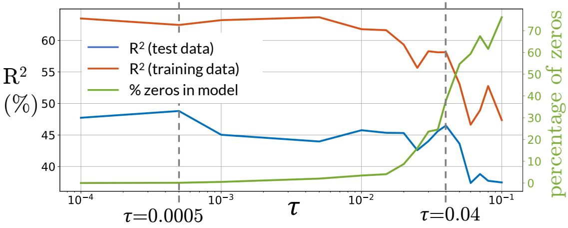
- **Note**: weighting $\|\theta_x^+\|_2^2 + \|\theta_x^-\|_2^2 + \|\theta_y^+\|_2^2 + \|\theta_y^-\|_2^2$ is **numerically better**, as ℓ_2-regularization is strongly convex for $\rho > 0$

INDUSTRIAL ROBOT BENCHMARK: RESULTS

- State $x \in \mathbb{R}^{12}$, f_x, f_y with $n_1^x = 24$ and $n_1^y = 12$ neurons, respectively, $\rho = 10^{-4}$
- Total number of training parameters: $\dim(\theta_x) + \dim(\theta_y) = 990$

INDUSTRIAL ROBOT BENCHMARK: RESULTS

- State $x \in \mathbb{R}^{12}$, f_x, f_y with $n_1^x = 24$ and $n_1^y = 12$ neurons, respectively, $\rho = 10^{-4}$
- Total number of training parameters: $\dim(\theta_x) + \dim(\theta_y) = 990$



(best R^2 in 5 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 $\approx$ 50 min on a single core of an Apple M1 Max CPU

INDUSTRIAL ROBOT BENCHMARK: RESULTS

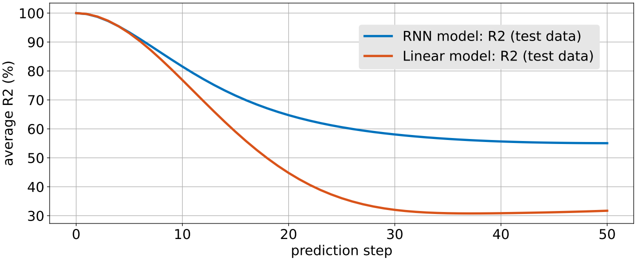
- **Open-loop simulation** errors ($\rho = 10^{-4}, \tau = 0.0005, n_1^x = 24, n_1^y = 12$):

	R^2 (training) RNN model	R^2 (test) RNN model	R^2 (training) linear model	R^2 (test) linear model
y_1	84.3099	74.3654	59.7335	59.9400
y_2	73.3438	53.2403	48.6032	31.9400
y_3	65.0838	47.0516	47.3231	24.1045
y_4	47.9524	46.2464	25.0829	21.6542
y_5	37.0665	34.3510	25.0987	24.8838
y_6	66.9417	37.5726	29.8516	31.5943
average	62.4497	48.8046	39.2822	**32.3528**

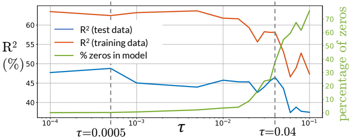
- More model parameters/smaller regularization leads to overfit training data

INDUSTRIAL ROBOT BENCHMARK: RESULTS

- Compute p-step ahead prediction $\hat{y}_{k+p|k}$, with hidden state $x_{k|k}$ estimated by an Extended Kalman Filter based on identified RNN model



- This is a more relevant indicator of model quality for MPC purposes than open-loop simulation error $\hat{y}_{k|0} - y_k$



- Compare **Adam** (Kingma, Ba, 2014) vs **L-BFGS-B**[1]:
  $(\tau = 0.04, \rho = 10^{-4}, n_1^x = 24, n_1^y = 12)$

method	best case criterion	average R^2 (training)	average R^2 (test)	# zeros	CPU time (s)
L-BFGS-B	R_2 (test)	58.13	**46.49**	375/990	3215
Adam		51.51	**47.31**	8/990	2511
L-BFGS-B	# zeros	54.34	**45.07**	520/990	3172
Adam		50.41	**41.99**	27/990	2518

Adam: tuned with learning rate exponentially decaying from 0.01 after 1000 steps, with decay rate 0.05.

- L-BFGS-B leads to **sparser models** than Adam with similar R^2-scores

[1]Best out of 5 runs, either based on the R_2 on test data or # zeros in θ_x, θ_y

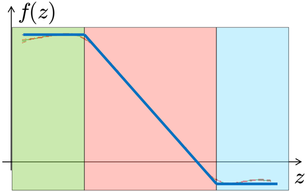
PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

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 - 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:

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), $v_{di} \in \{w_{di}^1, \dots, w_{di}^{m_i}\}$ (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**

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

PARC - PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

(Bemporad, 2022)

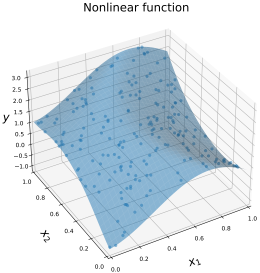
- New **Piecewise Affine Regression and Classification** (**PARC**) algorithm
- Training dataset:
  - **feature vector** $z \in \mathbb{R}^n$ (categorical features **one-hot encoded** in $\{0, 1\}$)
  - **target vector** $v_c \in \mathbb{R}^{m_c}$ (numeric), $v_{di} \in \{w_{di}^1, \dots, w_{di}^{m_i}\}$ (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**

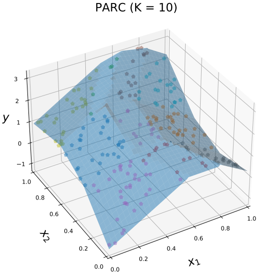
$$\Phi(z) = \omega^{j(z)} z + \gamma^{j(z)}, \qquad j(z) = \min\left\{\arg\max_{j=1,\dots,K}\{\omega^j z + \gamma^j\}\right\}$$

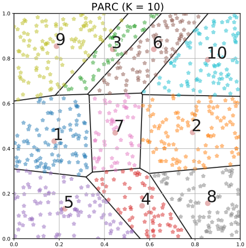
- Data reassigned to clusters based on weighted fit/PWL separation criterion
- PARC is a **block-coordinate descent** algorithm $\Rightarrow$ (local) convergence ensured

PARC - PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

- 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

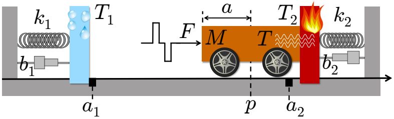






- Code: `pip install pyparc` **github.com/bemporad/PyPARC**

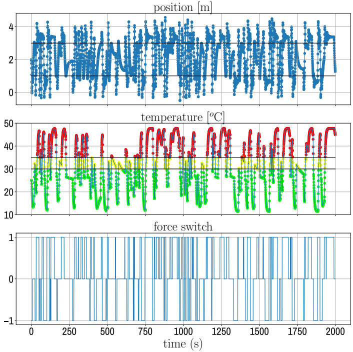
PARC - CART & BUMPERS EXAMPLE



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

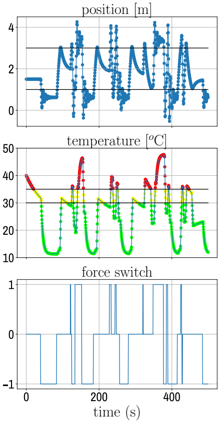
  Spring and viscous forces are **nonlinear**.

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

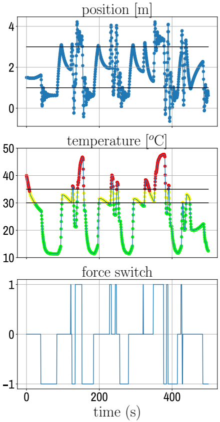


PARC - CART & BUMPERS EXAMPLE

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



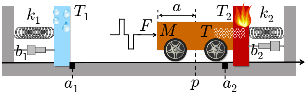
continuous-time system



discrete-time PWA model

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

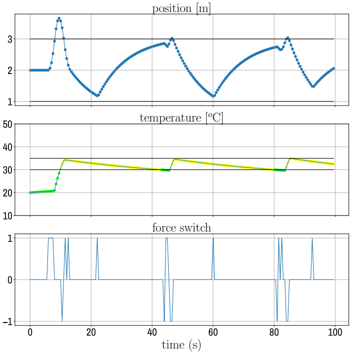
- MPC problem with prediction horizon $N = 9$:



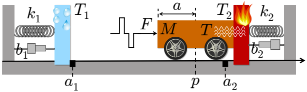
$$
\begin{aligned}
\min_{F_0,\ldots,F_{N-1}} \quad & \sum_{k=0}^{N-1} |c_k - 1| + 0.25|F_k| \\
\text{s.t.} \quad & F_k \in \{-\bar{F}, 0, \bar{F}\} \\
& \text{PWA model equations}
\end{aligned}
$$

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

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



PARC - CART & BUMPERS EXAMPLE

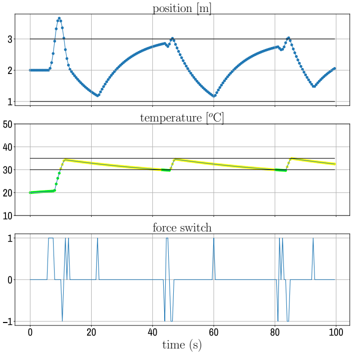


- MPC problem with prediction horizon $N = 9$:

$$\min_{F_0,\ldots,F_{N-1}} \quad \sum_{k=0}^{N-1} |c_k - 1| + 0.25|F_k|$$

$$\text{s.t.} \quad F_k \in \{-\bar{F}, 0, \bar{F}\}$$

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

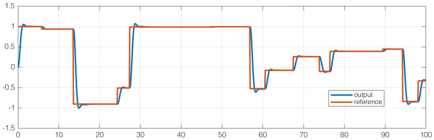
- The design depends on a vector x of **MPC parameters**

- 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**

$$\min_x f(x)$$

AUTO-TUNING - GLOBAL OPTIMIZATION ALGORITHMS

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

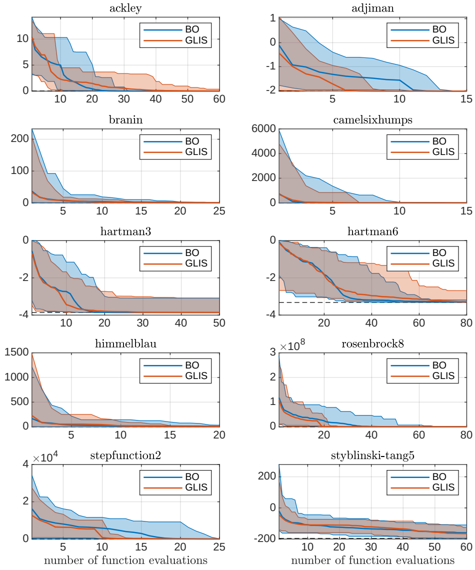
AUTO-TUNING - GLOBAL OPTIMIZATION ALGORITHMS

- Several derivative-free global optimization algorithms exist: (Rios, Sahidinis, 2013)
  - Lipschitzian-based partitioning techniques:
    - **DIRECT** (DIvide in RECTangles) (Jones, 2001)
    - Multilevel Coordinate Search (**MCS**) (Huyer, Neumaier, 1999)
  - Response surface methods
    - **Kriging** (Matheron, 1967), **DACE** (Sacks et al., 1989)
    - Efficient global optimization (**EGO**) (Jones, Schonlau, Welch, 1998)
    - **Bayesian optimization** (Brochu, Cora, De Freitas, 2010)
  - Genetic algorithms (**GA**) (Holland, 1975)
  - Particle swarm optimization (**PSO**) (Kennedy, 2010)
  - ...
- **GLIS** method - **radial basis function** surrogates + **inverse distance weighting** (Bemporad, 2020)

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

```
pip install glis
```

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

- Comparable performance
- GLIS is computationally lighter
- GLIS is more flexible

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, ...)

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:

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

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:

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

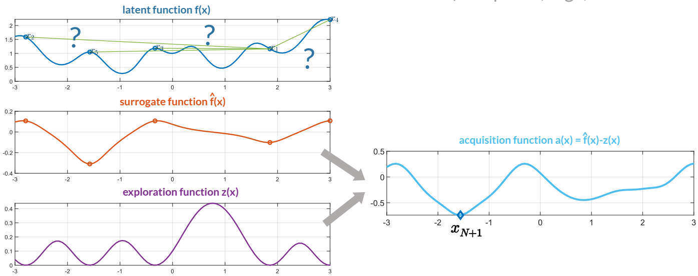
- We want to find a global optimum $x^\star$ (="better" than any other x)

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

- **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**

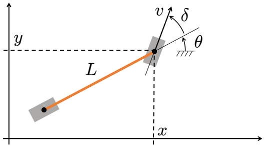
GLISp - GLIS based on **p**references (part of GLIS package)

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:

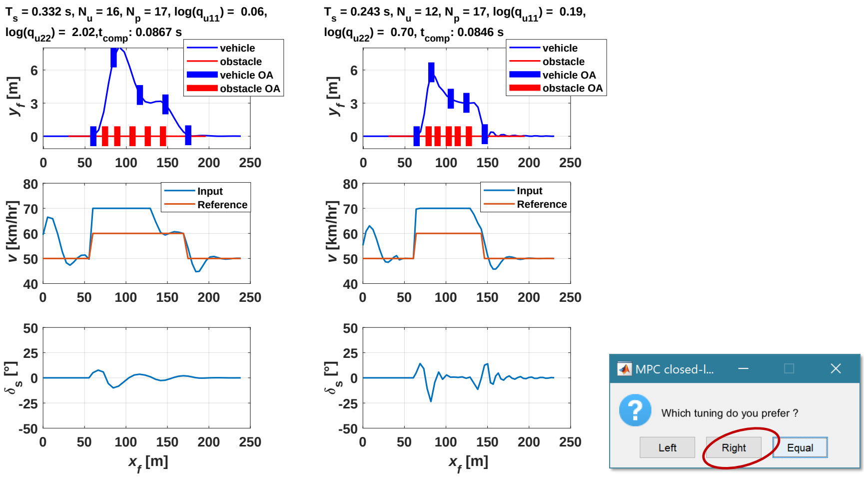
  "optimal obstacle avoidance", *"pleasant drive"*, *"CPU time small enough"*, …

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

- Latent function = calibrator's (unconscious) score

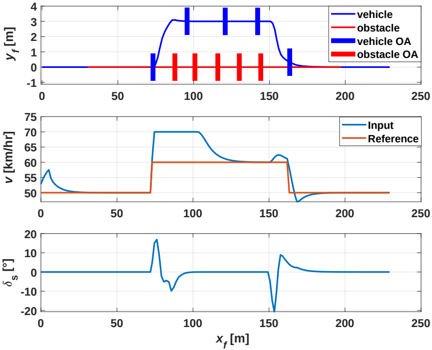
- 5 MPC parameters to tune:
  - **sampling time**
  - prediction and control **horizons**
  - **weights** on input increments $\Delta v, \Delta\delta$

- 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

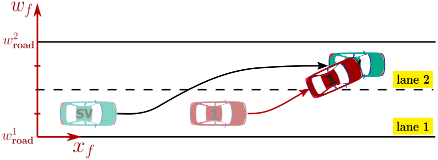
- **Note**: no need to define a closed-loop performance index explicitly!
- Extended to handle also **unknown constraints** (Zhu, Piga, Bemporad, 2021)

WORST-CASE SCENARIO DETECTION

WORST-CASE SCENARIO DETECTION

(Zhu, Bemporad, Kneissl, Esen, 2023)

- **Goal**: detect undesired closed-loop scenarios (=corner-cases)
- Let x = parameters defining the scenario (e.g., initial conditions, disturbances, ...)
- Critical scenario = vector x^* for which the closed-loop behavior is critical



- Critical scenario detection = find the worst combination x^* of scenario parameters by solving the global optimization problem

$$\min_x f(x)$$

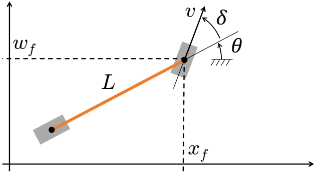
CORNER-CASE DETECTION: CASE STUDY

- **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)

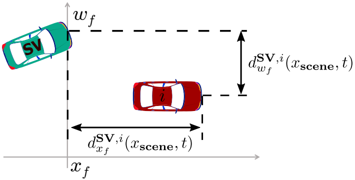
$$
\begin{aligned}
\dot{x}_f &= v\cos(\theta+\delta)\\
\dot{w}_f &= v\sin(\theta+\delta)\\
\dot{\theta} &= \frac{v\sin(\delta)}{L}
\end{aligned}
$$

(x_f, w_f) = front-wheel position



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

$$
\begin{aligned}
\min_{\ell\leq x\leq u} \quad & \sum_{i=1}^{k} d^{\text{SV},i}_{x_f,\text{critical}}(x) + d^{\text{SV},i}_{w_f,\text{critical}}(x)\\
\text{s.t.} \quad & \text{other constraints}
\end{aligned}
$$

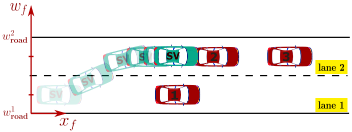


CORNER-CASE DETECTION: CASE STUDY

- **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

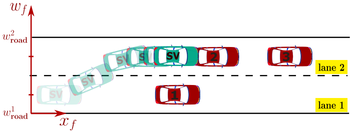


Ego car changes lane to avoid #1, but cannot brake fast enough to avoid #2

- **Logical scenario 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

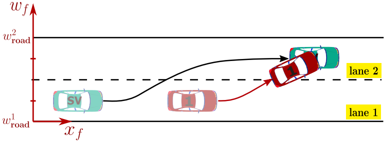


Ego car changes lane to avoid #1, but cannot brake fast enough to avoid #2

- **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

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 (=more sample-efficient, better performs tasks it wasn't trained for)

- 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 !