MODEL PREDICTIVE CONTROL

FROM BASICS TO LEARNING-BASED DESIGN

Alberto Bemporad

imt.lu/ab

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

ELO-X

ODYS Advanced Controls & Optimization

1st ELO-X Seasonal School - March 22, 2022

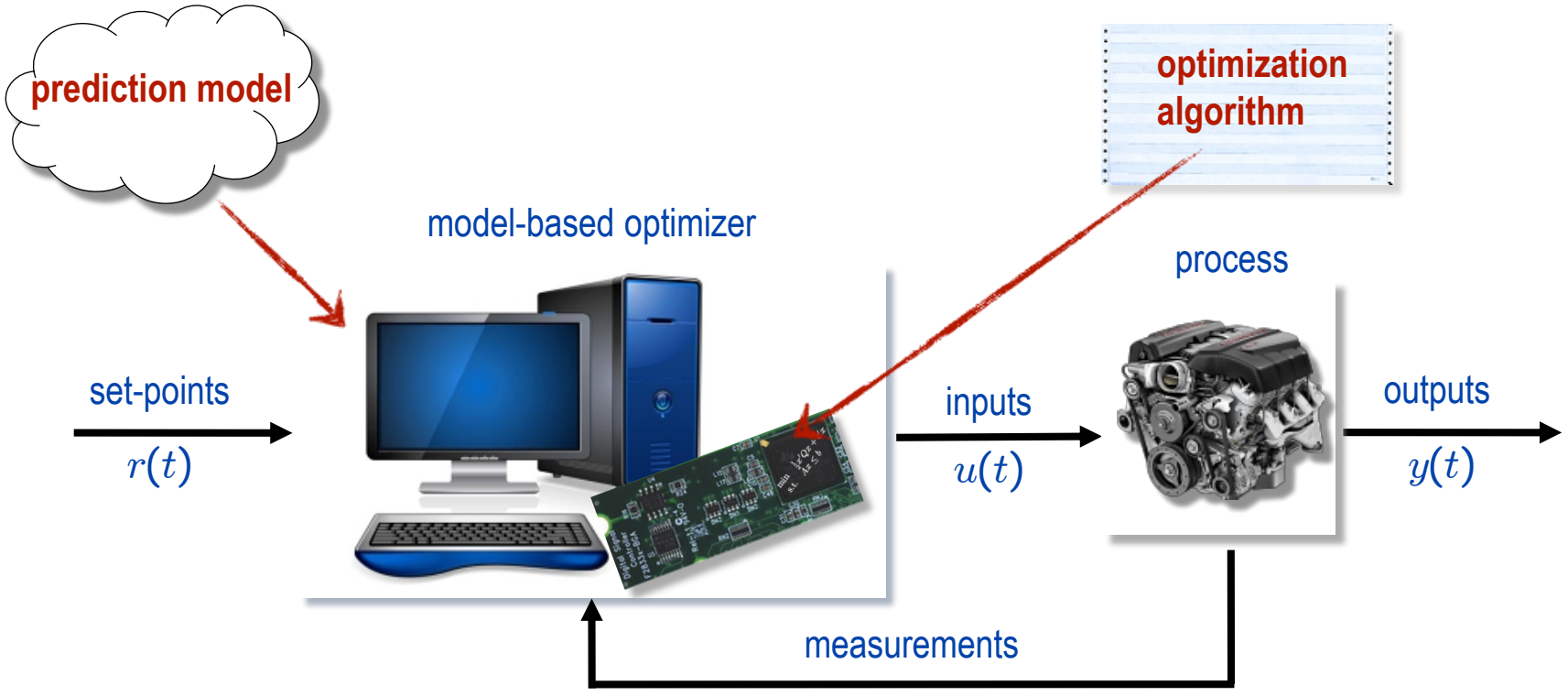
CONTENTS OF MY LECTURE

- Model predictive control (MPC): basic concepts
- Linear MPC and extensions to nonlinear MPC
- Embedded quadratic optimization
- Learning-based nonlinear MPC (feedforward and recurrent neural networks)
- Learning-based hybrid MPC (piecewise affine models)
- Active preference learning for MPC calibration

MODEL PREDICTIVE CONTROL: BASIC CONCEPTS

(extended slide set: `http://cse.lab.imtlucca.it/~bemporad/mpc_course.html`)

MODEL PREDICTIVE CONTROL (MPC)



Use a simplified dynamical **model** of the process to **predict** its likely future evolution and choose ~~the "best" control~~ a good action

MODEL PREDICTIVE CONTROL

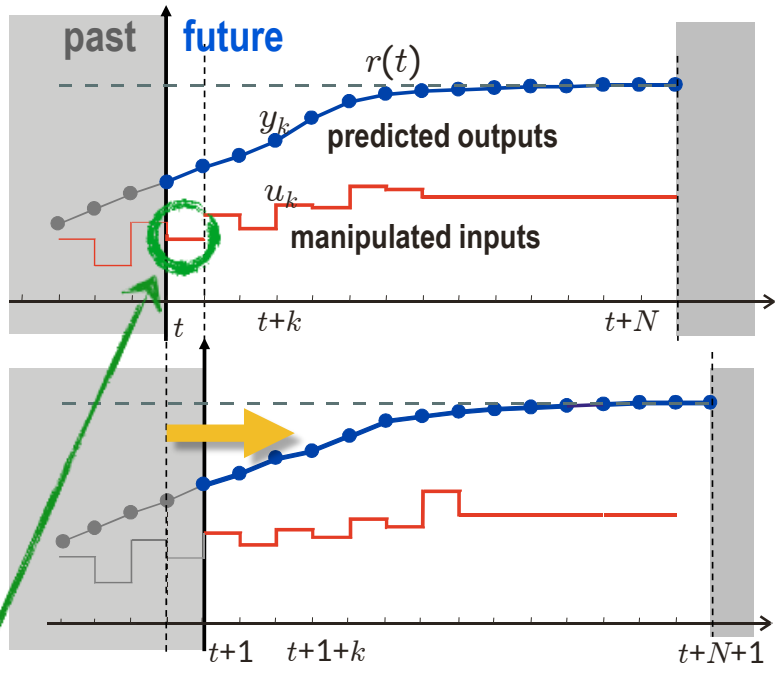
- **MPC problem**: find the best control sequence over a future horizon of N steps

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

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

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

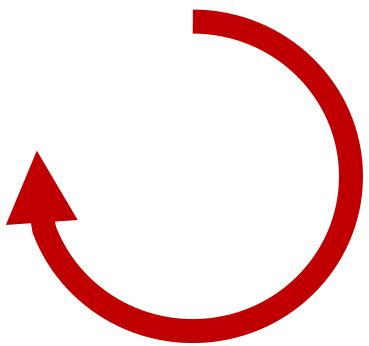
$$x_0 = x(t) \qquad \text{state feedback}$$



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 $u(t)$

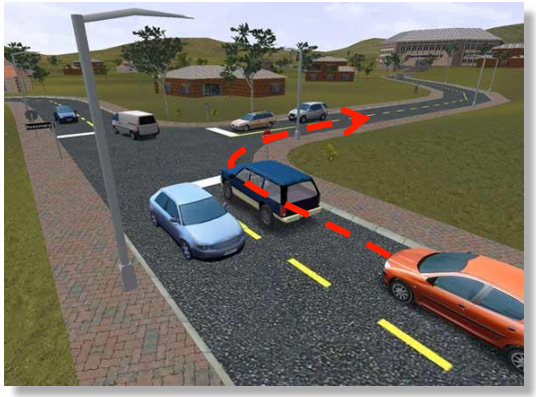
Repeat at all time steps t



DAILY-LIFE EXAMPLES OF MPC

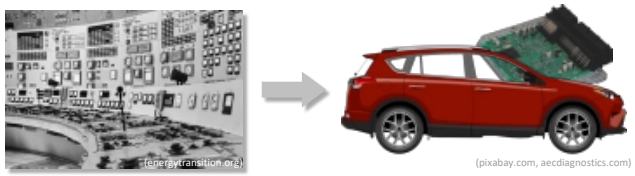
- MPC is like playing chess !

- You use MPC too when you drive !



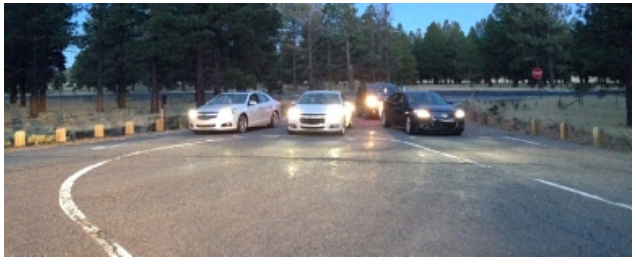
MPC IN INDUSTRY

- Conceived in the 60's (Rafal, Stevens, 1968) (Propoi, 1963)



- Used in the **process industries** since the 80's (Qin, Badgewell, 2003)
- Nowadays spreading to the automotive industry and other sectors
- MPC by **General Motors** and **ODYS** in high-volume production since 2018

  (Bemporad, Bernardini, Long, Verdejo, 2018)



First known mass production of MPC in the automotive industry

ODYS
Advanced Controls & Optimization

www.odys.it

MPC IN INDUSTRY

Table 2
The percentage of survey respondents indicating whether a control technology had demonstrated ("Current Impact") or was likely to demonstrate over the next five years ("Future Impact") high impact in practice.

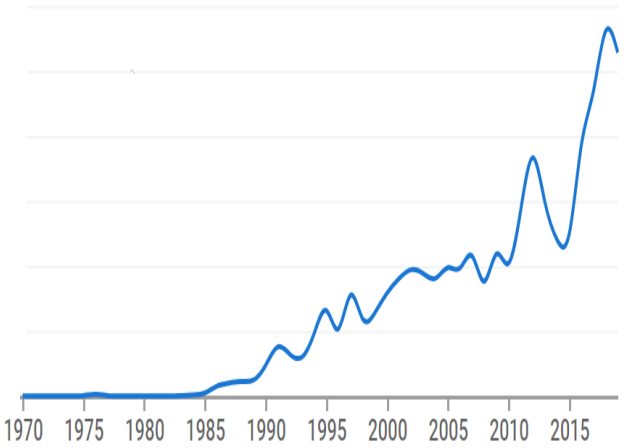
	Current Impact	Future Impact
Control Technology	%High	%High
PID control	91%	78%
System Identification	65%	72%
Estimation and filtering	64%	63%
Model-predictive control	62%	85%
Process data analytics	51%	70%
Fault detection and identification	48%	78%
Decentralized and/or coordinated control	29%	54%
Robust control	26%	42%
Intelligent control	24%	59%
Discrete-event systems	24%	39%
Nonlinear control	21%	42%
Adaptive control	18%	44%
Repetitive control	12%	17%
Hybrid dynamical systems	11%	33%
Other advanced control technology	11%	25%
Game theory	5%	17%

(Samad et al., 2020)

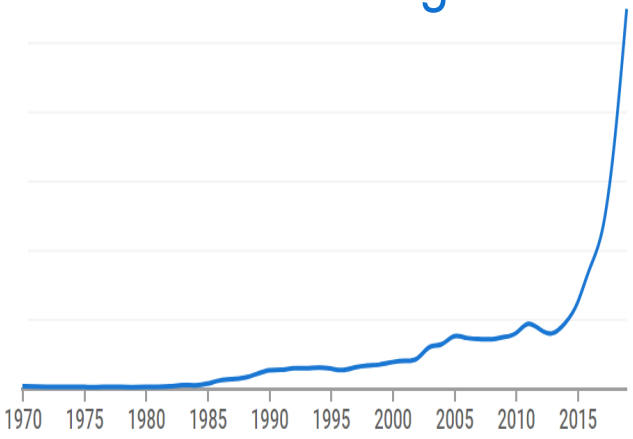
"As can be observed, MPC is clearly considered more impactful, and likely to be more impactful, vis-à-vis other control technologies, especially those that can be considered the "crown jewels" of control theory - robust control, adaptive control, and nonlinear control."

WORD TRENDS

model predictive control

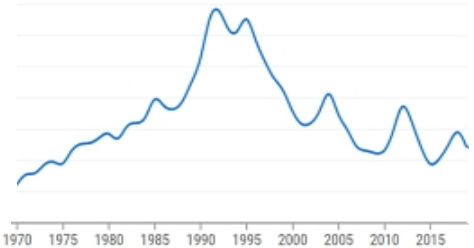


machine learning

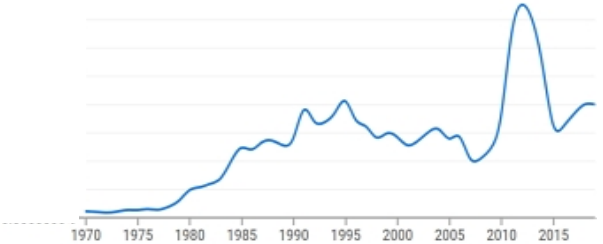


nonlinear control

system identification



PID control



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

MODEL PREDICTIVE CONTROL - THE BASICS

LINEAR MPC

- Linear prediction model: $\left\{ \begin{array}{rcl} x_{k+1} & = & Ax_k + Bu_k \\ y_k & = & Cx_k \end{array} \right.$ $\begin{array}{l} x \in \mathbb{R}^n \\ u \in \mathbb{R}^m \\ y \in \mathbb{R}^p \end{array}$

- Constrained optimal control problem (quadratic performance index):

$$\begin{array}{ll} \min\limits_{z} & x_N' P x_N + \sum\limits_{k=0}^{N-1} x_k' Q x_k + u_k' R u_k \\ \\ \text{s.t.} & u_{\min} \le u_k \le u_{\max},\ k = 0, \ldots, N-1 \\ & y_{\min} \le y_k \le y_{\max},\ k = 1, \ldots, N \end{array}$$

$$\begin{array}{rcl} R & = & R' \succ 0 \\ Q & = & Q' \succeq 0 \\ P & = & P' \succeq 0 \end{array} \qquad z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

LINEAR MPC

- Optimization problem (**condensed form**): $x_k = A^k x_0 + \sum_{i=0}^{k-1} A^i B u_{k-1-i}$

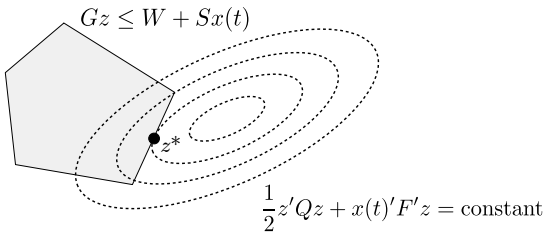
$$V(x_0) = \tfrac{1}{2} x_0' Y x_0 + \min_z \ \tfrac{1}{2} z' H z + x_0' F' z \quad \text{(quadratic objective)}$$

$$H = H' \succ 0$$

$$\text{s.t.} \quad Gz \le W + Sx_0 \quad \text{(linear constraints)}$$

convex Quadratic Program (QP)

- $z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \in \mathbb{R}^{Nm}$ is the optimization vector



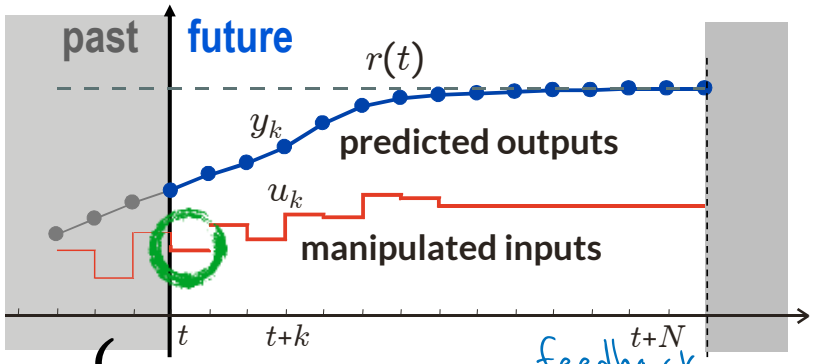
- QP matrices depend on chosen weights, model, and constraints

- **Alternative**: keep also $x_1, \ldots, x_N$ as optimization variables and the equality constraints $x_{k+1} = Ax_k + Bu_k$ (**non-condensed form**, which is **sparse**)

LINEAR MPC ALGORITHM

@ each sampling step t:

- Estimate the current state $x(t)$
- Get the solution $z^* = \begin{bmatrix} u_0^* \\ u_1^* \\ \vdots \\ u_{N-1}^* \end{bmatrix}$ of the QP



$$\begin{cases} \min\limits_{z} \quad \frac{1}{2} z' H z + \overbrace{x'(t) F' z}^{\text{feedback}} \\ \text{s.t.} \quad G z \le W + S \underbrace{x(t)}_{\text{feedback}} \end{cases}$$

- Apply only $u(t) = u_0^*$, discarding the remaining optimal inputs $u_1^*, \dots, u_{N-1}^*$
- **Unconstrained MPC**: $\overbrace{Hz + Fx(t)}^{\text{gradient}} = 0 \quad \Rightarrow \quad u(t) = -[I\, 0 \dots 0] H^{-1} F x(t)$ **linear state feedback**!

BASIC CONVERGENCE PROPERTIES

(Keerthi, Gilbert, 1988) (Bemporad, Chisci, Mosca, 1994)

- **Theorem**: Let the MPC law be based on

$$
\begin{aligned}
V^*(x(t)) = \min \quad & \sum_{k=0}^{N-1} x_k' Q x_k + u_k' R u_k \\
\text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k \\
& u_{\min} \le u_k \le u_{\max} \\
& y_{\min} \le Cx_k \le y_{\max} \\
& x_N = 0 \quad \leftarrow \text{“terminal constraint”}
\end{aligned}
$$

with $R, Q \succ 0, u_{\min} < 0 < u_{\max}, y_{\min} < 0 < y_{\max}$.

If the **optimization problem is feasible at time** $t = 0$ then

$$\lim_{t\to\infty} x(t) = 0, \quad \lim_{t\to\infty} u(t) = 0$$

and the constraints are satisfied at all time $t \ge 0$, for all $R, Q \succ 0$.

- Many more convergence and stability results exist (Mayne, 2014)

LINEAR MPC - TRACKING

- Objective: make the output $y(t)$ track a reference signal $r(t)$
- Let us parameterize the problem using the **input increments**

$$\Delta u(t) = u(t) - u(t-1)$$

- As $u(t) = u(t-1) + \Delta u(t)$ we need to extend the system with a new state $x_u(t) = u(t-1)$

$$\begin{cases} x(t+1) &= Ax(t) + Bu(t-1) + B\Delta u(t) \\ x_u(t+1) &= x_u(t) + \Delta u(t) \end{cases}$$

$$\begin{cases} \begin{bmatrix} x(t+1) \\ x_u(t+1) \end{bmatrix} &= \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x(t) \\ x_u(t) \end{bmatrix} + \begin{bmatrix} B \\ I \end{bmatrix} \Delta u(t) \\ y(t) &= \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ x_u(t) \end{bmatrix} \end{cases}$$

- Again a linear system with states $x(t), x_u(t)$ and input $\Delta u(t)$

LINEAR MPC - TRACKING

- Optimal control problem (quadratic performance index):

$$\begin{aligned}\min_{z} \quad & \sum_{k=0}^{N-1} \|W^y(y_{k+1} - r(t))\|_2^2 + \|W^{\Delta u}\Delta u_k\|_2^2 \\ & [\Delta u_k \triangleq u_k - u_{k-1}],\ u_{-1} = u(t-1) \\ \text{s.t.} \quad & u_{\min} \le u_k \le u_{\max},\ k = 0,\dots,N-1 \\ & y_{\min} \le y_k \le y_{\max},\ k = 1,\dots,N \\ & \Delta u_{\min} \le \Delta u_k \le \Delta u_{\max},\ k = 0,\dots,N-1\end{aligned}$$

$$z = \begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ \vdots \\ \Delta u_{N-1} \end{bmatrix} \text{ or } z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

weight $W^{(\cdot)}$ = diagonal matrix

$$\begin{aligned}\min_{z} \quad & J(z, x(t)) = \tfrac{1}{2} z' H z + [x'(t)\, r'(t)\, u'(t-1)] F' z \\ \text{s.t.} \quad & Gz \le W + S \begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix}\end{aligned}$$

convex Quadratic Program

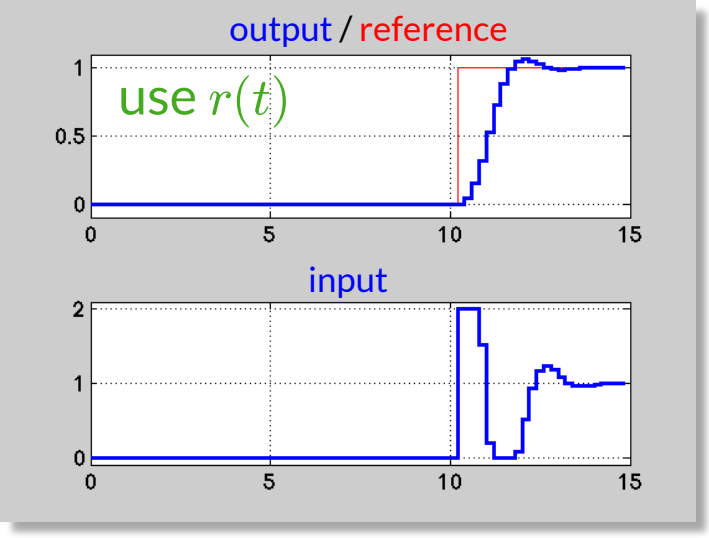
- Add the extra penalty $\|W^u(u_k - u_{\text{ref}}(t))\|_2^2$ to track **input references**
- Constraints may depend on $r(t)$, such as $e_{\min} \le y_k - r(t) \le e_{\max}$

ANTICIPATIVE ACTION (A.K.A. "PREVIEW")

$$\min_{\Delta U} \sum_{k=0}^{N-1} \|W^y(y_{k+1} - r(t+k))\|_2^2 + \|W^{\Delta u}\Delta u(k)\|_2^2$$

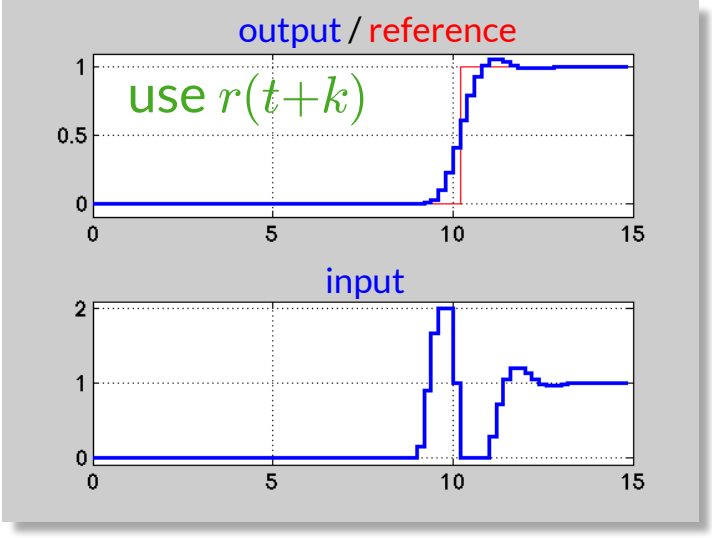
- Reference **not known** in advance (**causal**):

$$r_k \equiv r(t), \forall k = 0, \ldots, N-1$$



- Future refs (partially) **known** in advance (**anticipative action**):

$$r_k = r(t+k), \forall k = 0, \ldots, N-1$$



- Same for previewing **measured disturbances** $x_{k+1} = Ax_k + Bu_k + B_v v(t+k)$

OUTPUT INTEGRATORS AND OFFSET-FREE TRACKING

- Add constant unknown disturbances on measured outputs:

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

- Use the extended model to design a **state observer** (e.g., Kalman filter) that estimates both the state $\hat{x}(t)$ and disturbance $\hat{d}(t)$ from $y(t)$

- Why we get offset-free tracking in steady-state (intuitively):
  - the observer makes $C\hat{x}(t) + \hat{d}(t) \to y(t)$ **(estimation error)**
  - the MPC controller makes $C\hat{x}(t) + \hat{d}(t) \to r(t)$ **(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

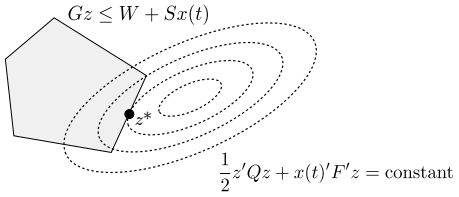
- See more on survey paper (Pannocchia, Gabiccini, Artoni, 2015)

EMBEDDED QUADRATIC OPTIMIZATION FOR MPC

EMBEDDED LINEAR MPC AND QUADRATIC PROGRAMMING

- MPC based on linear models requires solving a **Quadratic Program (QP)**

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z'Qz + x'(t)F'z + \frac{1}{2}x'(t)Yx(t) \\ \text{s.t.} \quad & Gz \leq W + Sx(t) \end{aligned} \qquad z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$$



ON MINIMIZING A CONVEX FUNCTION SUBJECT TO LINEAR INEQUALITIES

By E. M. L. BEALE

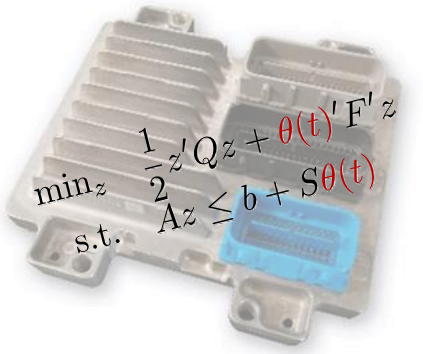
Admiralty Research Laboratory, Teddington, Middlesex

SUMMARY

THE minimization of a convex function of variables subject to linear inequalities is discussed briefly in general terms. Dantzig's Simplex Method is extended to yield finite algorithms for minimizing either a convex quadratic function or the sum of the t largest of a set of linear functions, and the solution of a generalization of the latter problem is indicated. In the last two sections a form of linear programming with random variables as coefficients is described, and shown to involve the minimization of a convex function.

(Beale, 1955)

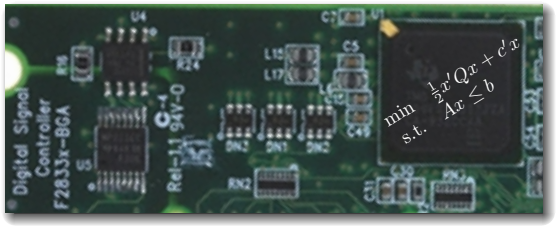
A rich set of good QP algorithms is available today



- Not all QP algorithms are suitable for **industrial embedded control**

MPC IN A PRODUCTION ENVIRONMENT

Key requirements for deploying MPC in production:



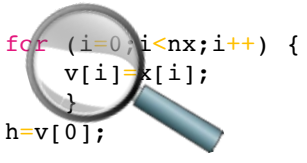
1. **speed (throughput)**
   - **worst-case** execution time less than sampling interval
   - also fast on **average** (to free the processor to execute other tasks)

2. limited **memory and CPU power** (e.g., 150 MHz / 50 kB)

3. **numerical robustness** (single precision arithmetic)

4. **certification** of worst-case execution time

5. **code simple enough** to be validated/verified/certified
   (library-free C code, easy to check by production engineers)

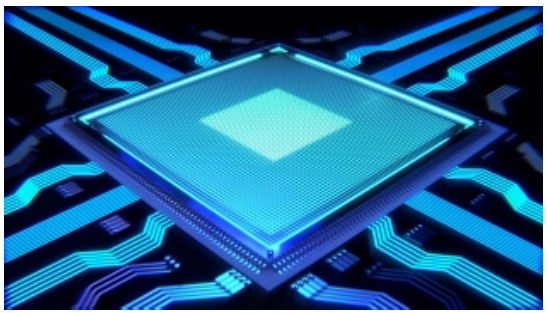


EMBEDDED SOLVERS IN INDUSTRIAL PRODUCTION

- Multivariable MPC controller
- Sampling frequency = 40 Hz (= 1 QP solved every 25 ms)
- Vehicle operating ≈1 hr/day for ≈360 days/year on average
- Controller running on 10 million vehicles

~520,000,000,000,000 QP/yr

and none of them should fail.



DUAL GRADIENT PROJECTION FOR QP

(Goldstein, 1964) (Levitin, Poljak, 1965) (Combettes, Waijs, 2005)

- Consider the strictly convex QP and its dual

$$\begin{array}{ll}\min & \frac{1}{2}z'Qz + x'F'z \\ \text{s.t.} & Gz \le W + Sx\end{array} \quad \Rightarrow \quad \begin{array}{ll}\min & \frac{1}{2}y'Hy + (Dx+W)'y \\ \text{s.t.} & y \ge 0\end{array}$$

with $H = GQ^{-1}G', D = S + GQ^{-1}F$. Take $L \ge \lambda_{\max}(H)$

- Apply **proximal gradient method** to dual QP:

$$y^{k+1} = \max\{y^k - \frac{1}{L}(Hy^k + Dx + W), 0\} \qquad y_0 = 0$$

- The primal solution is related to the dual solution by

$$z^k = -Q^{-1}(Fx + G'y^k)$$

- Convergence is slow: the initial error $f(z^0) - f(z^*)$ reduces as $1/k$

FAST GRADIENT PROJECTION FOR (DUAL) QP

(Nesterov, 1983) (Beck, Teboulle, 2008) (Patrinos, Bemporad, 2014)

- The **fast gradient method** is applied to solve the dual QP problem

$$\begin{array}{ll}\min\limits_{z} & \frac{1}{2}z'Qz + x'F'z \\ \text{s.t.} & Gz \le W + Sx\end{array}$$

$$\begin{aligned} K &= Q^{-1}G' \\ J &= Q^{-1}F \\ L &\ge \lambda_{\max}(GQ^{-1}G') \\ \beta_k &= \max\{\tfrac{k-1}{k+2}, 0\} \end{aligned}$$

$$\begin{aligned} w^k &= y^k + \beta_k(y^k - y^{k-1}) \\ z^k &= -Kw^k - Jx \\ s^k &= \tfrac{1}{L}Gz^k - \tfrac{1}{L}(W + Sx) \\ y^{k+1} &= \max\{w^k + s^k, 0\} \end{aligned}$$

```
while k<maxiter
  beta=max((k-1)/(k+2),0);
  w=y+beta*(y-y0);
  z=-(iMG*w+iMc);
  s=GL*z-bL;

  y0=y;


  % Termination
  if all(s<=epsGL)
    gapL=-w'*s;
    if gapL<=epsVL
      return
    end
  end

  y=w+s;
  k=k+1;
end
```

- Very **simple to code**

FAST GRADIENT PROJECTION FOR (DUAL) QP

- **Termination criteria**: when the following two conditions are met

$$\begin{array}{rcl} s_i^k & \leq & \frac{1}{L}\epsilon_G,\ i=1,\dots,m \\ -(w^k)'s^k & \leq & \frac{1}{L}\epsilon_f \end{array}$$

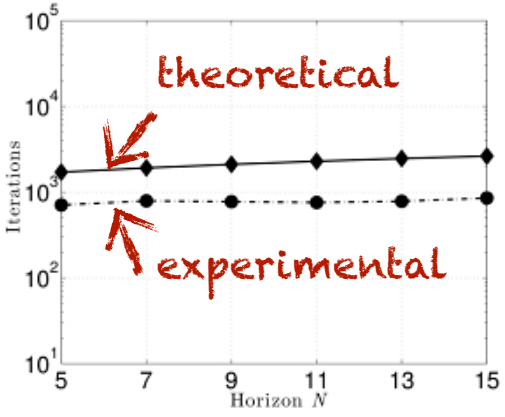
primal feasibility

optimality

the solution $z^k = -Kw^k - Jx$ satisfies $G_i z^k - W_i - S_i x \leq \epsilon_G$ and, if $w^k \geq 0$,

$$f(z^k) - f(z^*) \leq f(z^k) - \underbrace{q(w^k)}_{\text{dual fcn}} = -(w^k)'s^k L \leq \epsilon_f$$

- Convergence rate: $f(x^k) - f(x^*) \leq \dfrac{2L}{(k+2)^2}\|z_0 - z^*\|_2^2$



- Tight bounds on maximum number of iterations
- Can be useful to warm-start active-set methods (Bemporad, Paggi, 2015)
- Extended to **mixed-integer quadratic programming (MIQP)** (Naik, Bemporad, 2017)

ADMM

(Gabay, Mercier, 1976) (Glowinski, Marrocco, 1975) (Douglas, Rachford, 1956) (Boyd et al., 2010)

- Alternating Directions Method of Multipliers for QP

$$\begin{array}{ll}\min & \frac{1}{2}z'Qz + c'z \\ \text{s.t.} & \ell \le Az \le u\end{array}$$

$$\begin{aligned} z^{k+1} &= -(Q+\rho A'A)^{-1}(\rho A'(v^k - s^k) + c) \\ s^{k+1} &= \min\{\max\{Az^{k+1} + v^k, \ell\}, u\} \\ v^{k+1} &= v^k + Az^{k+1} - s^{k+1} \end{aligned}$$

```
while k<maxiter
  k=k+1;
  z=-iM*(c+A'*(rho*(v-s)));
  Az=A*z;
  s=max(min(Az+v,u),ell);
  v=v+Az-s;
end
```

(7 lines EML code)
(≈40 lines of C code)

ρv = dual vector

- Matrix $(Q + \rho A'A)$ must be nonsingular
- The factorization of matrix $(Q + \rho A'A)$ can be done at start and cached
- Very **simple to code**. Sensitive to matrix **scaling** (as gradient projection)
- Used in many applications (control, signal processing, machine learning)

REGULARIZED ADMM FOR QUADRATIC PROGRAMMING

(Stellato, Banjac, Goulart, Bemporad, Boyd, 2020)

- Robust "regularized" ADMM iterations:

$$\begin{aligned} z^{k+1} &= -(Q + \rho A^T A + \epsilon I)^{-1}(c - \epsilon z_k + \rho A^T(v^k - z^k)) \\ s^{k+1} &= \min\{\max\{Az^{k+1} + v^k, \ell\}, u\} \\ v^{k+1} &= v^k + Az^{k+1} - s^{k+1} \end{aligned}$$

- Works for any $Q \succeq 0$, A, and choice of $\epsilon > 0$
- **Simple** to code, **fast**, and **robust**
- Only needs to factorize $\begin{bmatrix} Q + \epsilon I & A' \\ A & -\frac{1}{\rho} I \end{bmatrix}$ once
- Implemented in free **osQP solver**

  (Python interface: ≈ 1,700,000 downloads)

  `http://osqp.org`

- Extended to solve **mixed-integer quadratic programming** problems

  (Stellato, Naik, Bemporad, Goulart, Boyd, 2018)

ODYS QP SOLVER

- General purpose QP solver designed for **industrial production**

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z'Qz + c'z \\ \text{s.t.} \quad & b_\ell \leq Az \leq b_u \\ & \ell \leq z \leq u \\ & Ez = f \end{aligned}$$

- Implements a **proprietary** state-of-the-art method for QP
- Completely written in **ANSI-C** and **MISRA-C 2012** compliant
- **Fast**, **robust** (also in single precision), **low-memory** requirements
- **optimized version for MPC** available (≈ 50% faster)
- Licensed to several automotive OEMs and Tier-1 suppliers
- **Certifiable** execution time

odys.it/qp

PRIMAL-DUAL INTERIOR-POINT METHOD FOR QP

(Nocedal, Wright, 2006) (Gondzio, Terlaki, 1994)

- The Karush-Kuhn-Tucker(KKT) optimality conditions for the convex QP

$$\begin{array}{ll} \min_x & \frac{1}{2}x'Qx + c'x \\ \text{s.t.} & Ax \le b \\ & Ex = f \end{array} \qquad Q = Q' \succeq 0$$

are

$$\begin{array}{rcccl} r_Q & = & Qx + c + E'y + A'z & = & 0 \\ r_E & = & Ex - f & = & 0 \\ r_A & = & Ax + s - b & = & 0 \\ r_S & = & [z_1 s_1 \ \dots \ z_m s_m]' & = & 0 \\ z, s & \ge & 0 & & \end{array}$$

x = primal vars
y = dual vars (eq. constr.)
s = slacks (ineq. constr.)
z = dual vars (ineq. constr.)

- In a nutshell, **interior-point** methods use Newton's method with line search to solve the above nonlinear system of equations

- The complementary slackness constraint is replaced by $z_i s_i = \mu$ and $\mu \to 0$

PRIMAL-DUAL INTERIOR-POINT METHOD FOR QP

(Nocedal, Wright, 2006) (Gondzio, Terlaki, 1994)

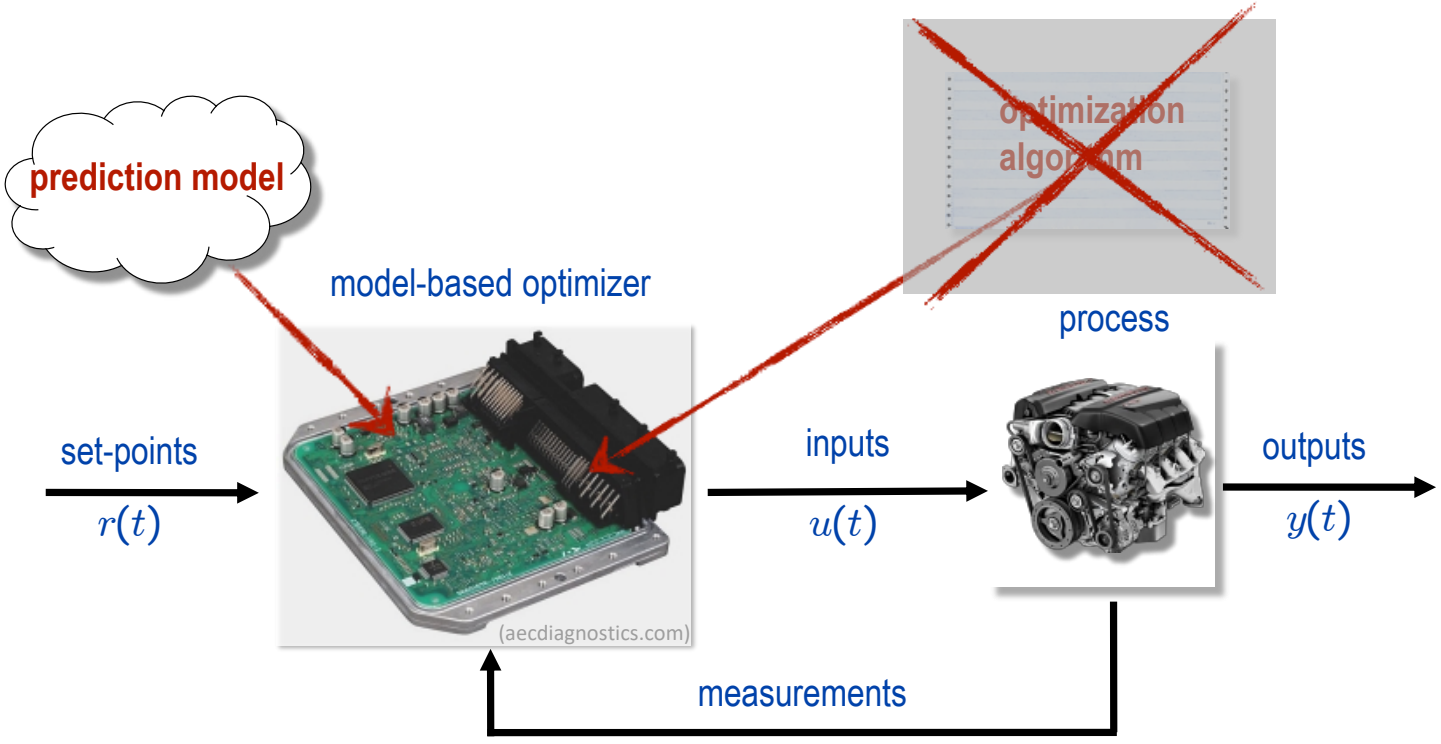
- Each interior-point iteration requires solving a linear system of the form

$$\begin{bmatrix} Q & E' & A' & 0 \\ E & 0 & 0 & 0 \\ A & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_Q \\ -r_E \\ -r_A \\ -r_S \end{bmatrix}$$

$$\begin{aligned} Z &= \operatorname{diag} z \\ S &= \operatorname{diag} s \end{aligned}$$

- In MPC the **structure** $x_{k+1} = Ax_k + Bu_k$ can be heavily exploited to factorize/solve the linear systems efficiently (Rao, Wright, Rawlings, 1998) (Wright, 2018)
- IP provides good solutions within 10-15 IP iterations (**usually** ...).
- Linear systems tends to become ill-conditioned at convergence
- IP usually faster for sparse and large QPs (say >500 vars & constraints)

MPC WITHOUT ON-LINE QP



- Can we implement constrained linear MPC **without an on-line QP solver** ?

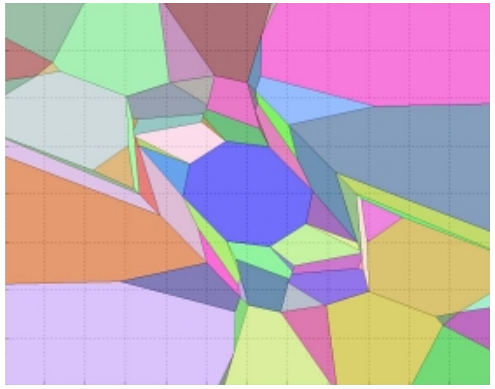
YES !

EXPLICIT MODEL PREDICTIVE CONTROL

- **Continuous** & **piecewise affine** solution of strictly convex multiparametric QP

$$z^*(x) = \arg\min_z \quad \tfrac{1}{2} z'Qz + x'F'z$$
$$\text{s.t.} \quad Gz \le W + Sx$$

(Bemporad, Morari, Dua, Pistikopoulos, 2002)



- Corollary: **linear MPC is continuous & piecewise affine** !

$$z^* = \begin{bmatrix} \mathbf{u_0} \\ u_1 \\ \vdots \\ u^*_{N-1} \end{bmatrix}$$

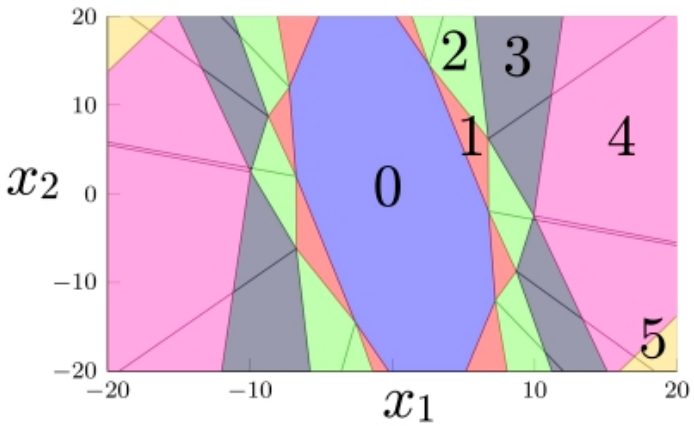
$$u_0^*(x) = \begin{cases} F_1 x + g_1 & \text{if} \quad H_1 x \le K_1 \\ \vdots & \vdots \\ F_M x + g_M & \text{if} \quad H_M x \le K_M \end{cases}$$

- New mpQP solver based on NNLS available (Bemporad, 2015) and included in **MPC Toolbox** since R2014b (Bemporad, Morari, Ricker, 1998-today)

Is explicit MPC better than on-line QP (=implicit MPC) ?

COMPLEXITY CERTIFICATION FOR ACTIVE-SET QP SOLVERS

- **Result**: The **number of iterations** to solve the QP via a dual active-set method is a **piecewise constant function** of the parameter x



(Cimini, Bemporad, 2017)

We can **exactly** quantify how many iterations (flops) the QP solver takes in the worst-case !

- Examples (from MPC Toolbox):

	inverted pendulum	DC motor	nonlinear demo	AFTI F16
Explicit MPC				
max flops	3382	1689	9184	16434
max memory (kB)	55	30	297	430
Implicit MPC				
max flops	3809	2082	7747	7807
sqrt	27	9	37	33
max memory (kB)	15	13	20	16

- QP certification algorithm currently used in industrial production projects

FROM LINEAR TO NONLINEAR MPC

LINEAR TIME-VARYING MODELS

- **Linear Time-Varying (LTV)** model

$$\begin{cases} x_{k+1} &= A_k(t)x_k + B_k(t)u_k \\ y_k &= C_k(t)x_k \end{cases}$$

- At each time t the model can also change over the prediction horizon k
- Possible measured disturbances are embedded in the model
- On-line optimization is still a QP

$$\begin{aligned} \min_z \quad & \frac{1}{2}z'H(t)z + \begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix}' F(t)'z \\ \text{s.t.} \quad & G(t)z \le W(t) + S(t)\begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix} \end{aligned}$$

- The QP matrices cannot be constructed offline

LINEARIZING A NONLINEAR MODEL

- LTV models can be obtained by linearizing a **nonlinear model**

$$\begin{cases} \frac{dx_c(t)}{dt} &= f(x_c(t), u_c(t)) \\ y_c(t) &= g(x_c(t)) \end{cases}$$

- At time t, consider the **nominal trajectory**

$$U = \{\bar{u}_c(t), \bar{u}_c(t+T_s), \ldots, \bar{u}_c(t+(N-1)T_s)\}$$

For example U = shifted previous sequence optimized by MPC @$t-1$

- **Integrate** the model from $\bar{x}_c(t)$ and get nominal state/output trajectories

$$X = \{\bar{x}_c(t), \bar{x}_c(t+T_s), \ldots, \bar{x}_c(t+(N-1)T_s)\}$$
$$Y = \{\bar{y}_c(t), \bar{y}_c(t+T_s), \ldots, \bar{y}_c(t+(N-1)T_s)\}$$

For example $\bar{x}_c(t) =$ current state

LINEARIZING A NONLINEAR MODEL

- **Linearize** the nonlinear model around the nominal states and inputs:

$$\frac{dx_c}{dt} = f(x_c,u_c) \approx \underbrace{f(\bar{x}_c,\bar{u}_c)}_{\frac{d\bar{x}_c}{dt}} + \underbrace{\left.\frac{\partial f}{\partial x_c}\right|_{\bar{x}_c,\bar{u}_c}}_{\text{Jacobian matrix } A_c}(x_c-\bar{x}_c) + \underbrace{\left.\frac{\partial f}{\partial u_c}\right|_{\bar{x}_c,\bar{u}_c}}_{\text{Jacobian matrix } B_c}(u_c-\bar{u}_c)$$

$$y = g(x_c) \approx \underbrace{g(\bar{x}_c)}_{\bar{y}_c} + \underbrace{\left.\frac{\partial g}{\partial x_c}\right|_{\bar{x}_c,\bar{u}_c}}_{\text{Jacobian matrix } C}(x_c-\bar{x}_c)$$

- Define $x \triangleq x_c - \bar{x}_c, u \triangleq u_c - \bar{u}_c, y \triangleq y_c - \bar{y}_c$ and get the linear system

$$\frac{dx}{dt} = A_c x + B_c u \qquad y = Cx$$

- Convert linear model to **discrete-time** and get matrices (A_k, B_k, C_k)

- **Alternative**: compute (A_k, B_k, C_k) (a.k.a. **sensitivities**) during integration

FROM LTV-MPC TO NONLINEAR MPC

- How to use the LTV-MPC machinery to handle nonlinear MPC ?

- **Key idea**: Solve a **sequence of LTV-MPC** problems at each time t

For $h = 0$ to $h_{\max} - 1$ do:

1. **Simulate** from $x(t)$ with inputs U_h and get state trajectory X_h
2. **Linearize** around (X_h, U_h) and **discretize** in time
3. Get U^*_{h+1} = **QP solution** of corresponding LTV-MPC problem
4. **Line search**: find optimal step size $\alpha_h \in (0, 1]$;
5. Set $U_{h+1} = (1 - \alpha_h)U_h + \alpha_h U^*_{h+1}$;

Return solution $U_{h_{\max}}$

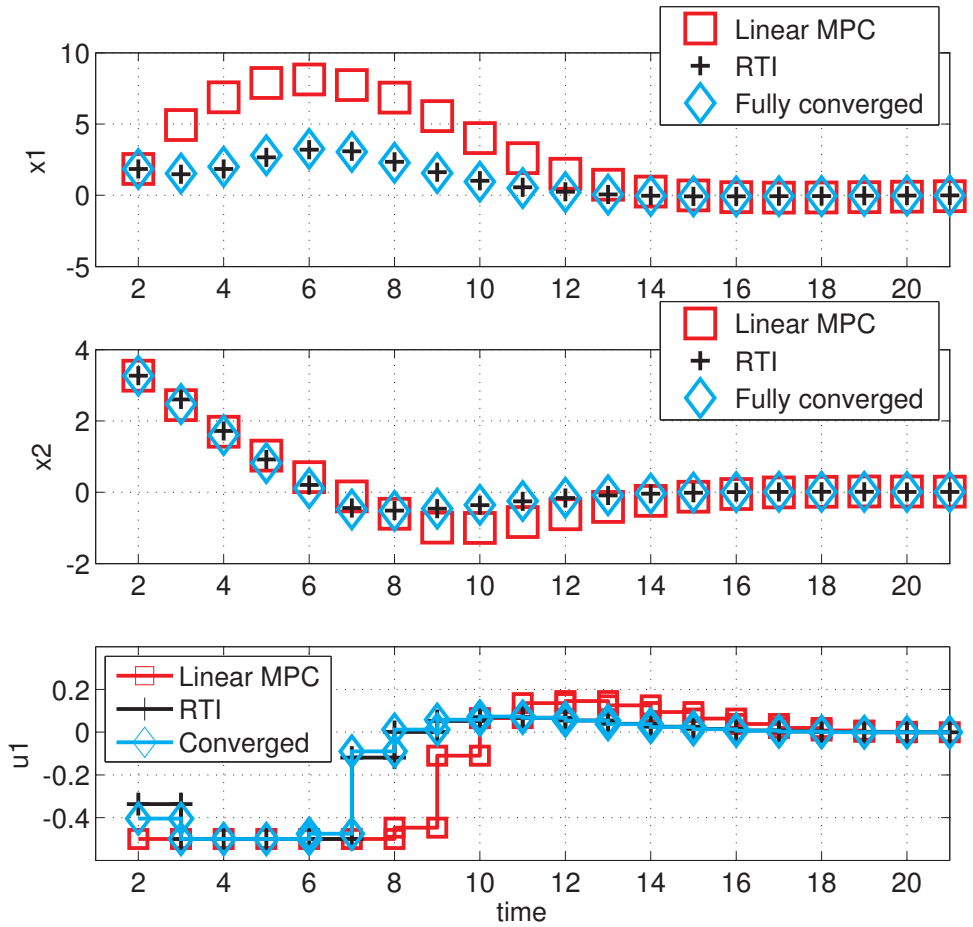
- Special case: just solve one iteration with $\alpha = 1$ (a.k.a. **Real-Time Iteration**)

  (Diehl, Bock, Schloder, Findeisen, Nagy, Allgower, 2002) = LTV-MPC

NONLINEAR MPC

(Gros, Zanon, Quirynen, Bemporad, Diehl, 2020)

- Example



OUTPUT FEEDBACK - EXTENDED KALMAN FILTER

- For **state estimation**, an **Extended Kalman Filter** (EKF) can be used based on the same nonlinear model (with additional noise)

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

- **measurement update**:

$$\begin{aligned} C(k) &= \frac{\partial g}{\partial x}(\hat{x}_{k|k-1}) \\ M(k) &= P(k|k-1)C(k)'[C(k)P(k|k-1)C(k)' + R(k)]^{-1} \\ \text{consumed by MPC} \rightarrow \hat{x}(k|k) &= \hat{x}(k|k-1) + M(k)\,(y(k) - g(\hat{x}(k|k-1))) \\ P(k|k) &= (I - M(k)C(k))P(k|k-1) \end{aligned}$$

- **time update**:

$$\hat{x}(k+1|k) = f(\hat{x}(k|k), u(k))$$

$$A(k) = \frac{\partial f}{\partial x}(\hat{x}_{k|k}, u(k), E[\xi(k)]),\ G(k) = \frac{\partial f}{\partial \xi}(\hat{x}_{k|k}, u(k), E[\xi(k)])$$

$$P(k+1|k) = A(k)P(k|k)A(k)' + G(k)Q(k)G(k)'$$

ODYS EMBEDDED MPC TOOLSET

- **ODYS Embedded MPC** is a software toolchain for design and deployment of MPC solutions in industrial production

ODYS
Advanced Controls & Optimization

- Support for **linear & nonlinear MPC** and **extended Kalman filtering**
- Extremely flexible, all MPC parameters can be changed at runtime
- Integrated with MPC-optimized version of **ODYS QP Solver**
- Library-free C code, **MISRA-C 2012 compliant**, supports also single precision
- Currently used worldwide by several automotive OEMs in R&D and production
- **MPC Toolbox Plugin** to easily import NL-MPC projects from MPC Toolbox
- **ODYS Deep Learning** supports **neural networks** as prediction models

odys.it/embedded-mpc

LEARNING-BASED NONLINEAR MPC

MACHINE LEARNING (ML)

- Good **mathematical foundations** from artificial intelligence, statistics, optimization

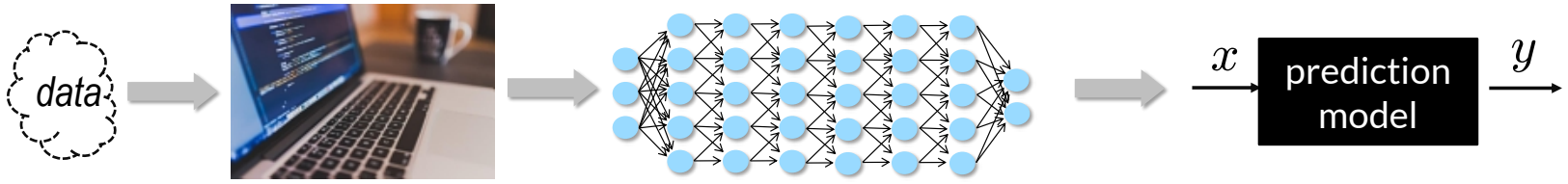
- **Works very well** in practice (despite training is most often a nonconvex optimization problem ...)

- Used in myriads of **very diverse application domains**

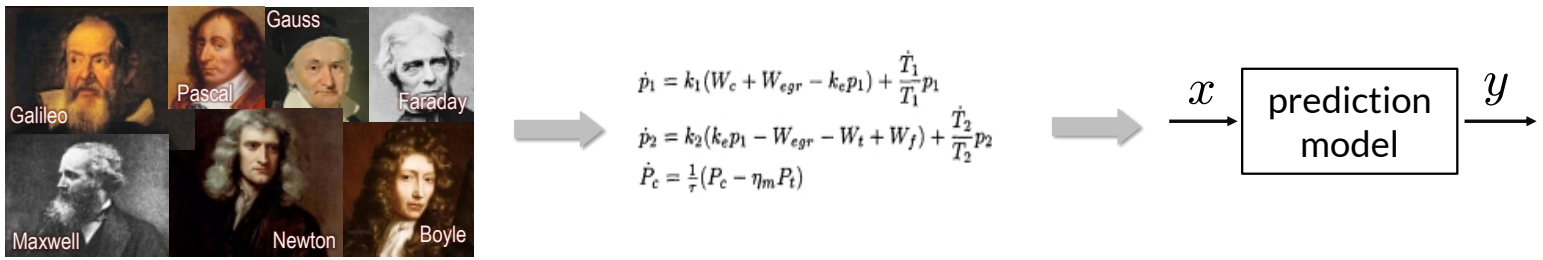
- Availability of excellent open-source **software tools** also explains success
  `scikit-learn`, **`TensorFlow/Keras`**, **`PyTorch`**, **`JAX`**, **`Flux.jl`**, ... python julia

CONTROL-ORIENTED NONLINEAR MODELS

- **Black-box modeling**: purely data-driven. Use training data to fit a prediction model that can explain them



- **Physics-based modeling**: use physical principles to create a prediction model (e.g.: weather forecast, chemical reaction, mechanical laws, ...)



- **Gray-box modeling** is a mix of the two. It 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

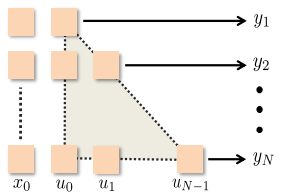
  (Hunt et al., 1992) (Suykens, Vandewalle, De Moor, 1996)

- **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)

- Alternative for MPC: learn entire prediction (Masti, Smarra, D'Innocenzo, Bemporad, 2020)

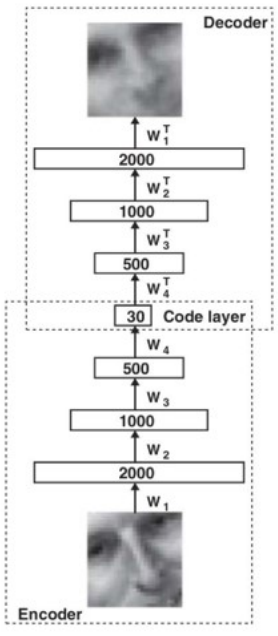
$$y_{t+k} = h_k(x_t, u_t, \ldots, u_{t+k-1}),\ k = 1, \ldots, N$$



- **Recurrent neural networks** are more appropriate for accurate open-loop predictions, but more difficult to train (see later ...)

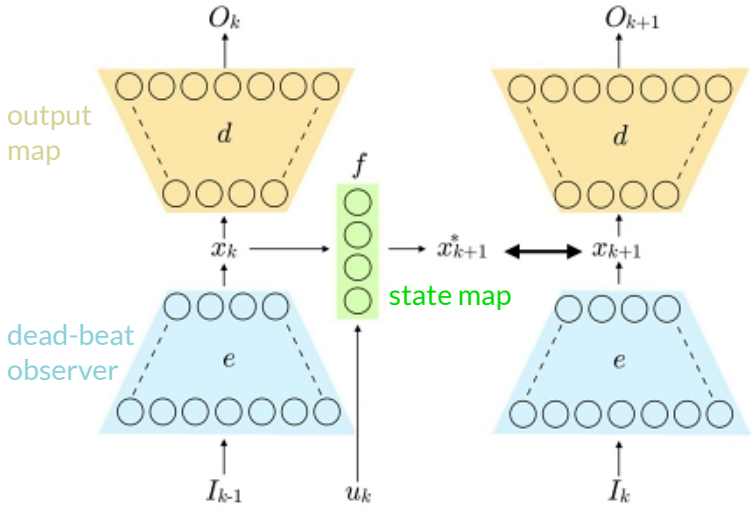
NONLINEAR STATE-SPACE MODELS VIA AUTOENCODERS

- **Idea**: use **autoencoders** and artificial neural networks to learn a **nonlinear state-space model** of **desired order** from input/output data



ANN with hourglass structure

(Hinton, Salakhutdinov, 2006)



$$O_k = [y'_k \ \dots \ y'_{k-m}]'$$

$$I_k = [y'_k \ \dots \ y'_{k-n_a+1} \ u'_k \ \dots \ u'_{k-n_b+1}]'$$

(Masti, Bemporad, 2021)

- **Quasi-LPV** structure for MPC: set $(A_{ij}, B_{ij}, C_{ij}$ = feedforward NNs)

$$x_{k+1} = A(x_k, u_k)\begin{bmatrix} x_k \\ 1 \end{bmatrix} + B(x_k, u_k)u_k$$

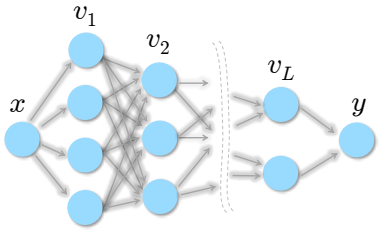
$$y_k = C(x_k, u_k)\begin{bmatrix} x_k \\ 1 \end{bmatrix}$$

LEARNING NEURAL NETWORK MODELS FOR CONTROL

TRAINING FEEDFORWARD NEURAL NETWORKS

- **Feedforward neural network** model:

$$y_k = f_y(x_k, \theta) = \begin{cases} v_{1k} &= A_1 x_k + b_1 \\ v_{2k} &= A_2 f_1(v_{1k}) + b_2 \\ \vdots & \vdots \\ v_{Lk} &= A_{L_y} f_{L-1}(v_{(L-1)k}) + b_L \\ \hat{y}_k &= f_L(v_{Lk}) \end{cases}$$

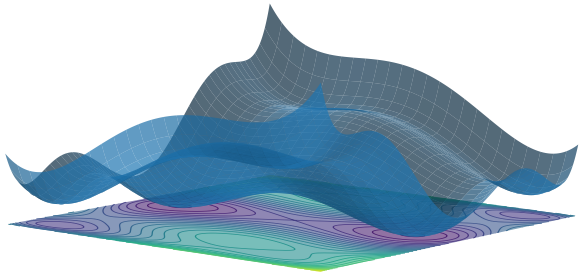


$$\theta = (A_1, b_1, \ldots, A_L, b_L)$$

Examples: x_k = measured state, or $x_k = (y_{k-1}, \ldots, y_{k-n_a}, u_{k-1}, \ldots, u_{k-n_b})$

- **Training problem**: given a dataset $\{x_0, y_0, \ldots, x_{N-1}, y_{N-1}\}$ solve

$$\min_{\theta} r(\theta) + \sum_{k=0}^{N-1} \ell(y_k, f(x_k, \theta))$$



- It is a nonconvex, unconstrained, nonlinear programming problem that can be solved by **stochastic gradient descent**, **quasi-Newton** methods, ... and **EKF** !

TRAINING FEEDFORWARD NEURAL NETWORKS BY EKF

(Singhal, Wu, 1989) (Puskorius, Feldkamp, 1994)

- **Key idea**: treat parameter vector θ of the feedforward neural network as a **constant state**

$$\begin{cases} \theta_{k+1} &= \theta_k + \eta_k \\ y_k &= f(x_k, \theta_k) + \zeta_k \end{cases}$$

  and use EKF to estimate θ_k **on line** from a streaming dataset $\{x_k, y_k\}$

- Ratio $\mathrm{Var}[\eta_k]/\mathrm{Var}[\zeta_k]$ is related to the **learning-rate**

- Initial matrix $(P_{0|-1})^{-1}$ is related to **quadratic regularization** on θ

- Implemented in **ODYS Deep Learning** library

- Extended to rather **arbitrary convex loss functions/regularization** terms

  (Bemporad, 2021 - `https://arxiv.org/abs/2111.02673`)

RECURRENT NEURAL NETWORKS

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

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

f_x, f_y = feedforward neural network

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

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

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

TRAINING RNNS ONLINE BY EKF

(Bemporad, 2021 - https://arxiv.org/abs/2111.02673)

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

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

- 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

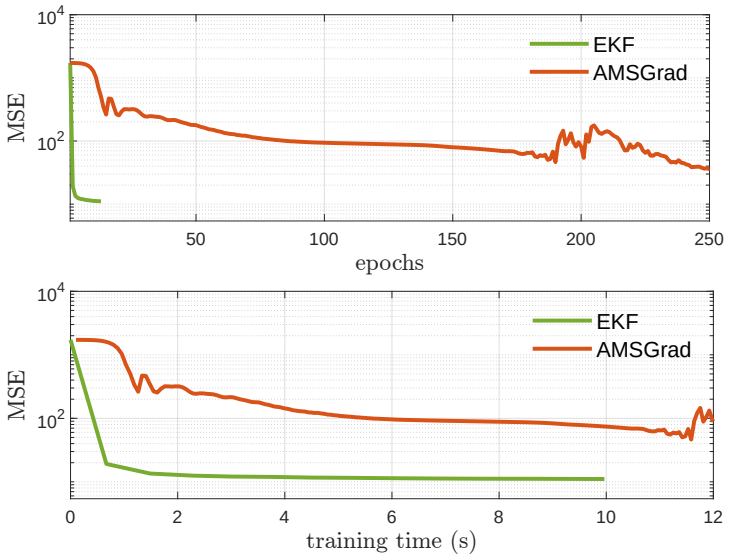
- Can handle **general smooth strictly convex** loss functions/regularization terms

- 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

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

TRAINING RNNS BY EKF - EXAMPLES

- Dataset: 3499 I/O data of **magneto-rheological fluid damper** (Wang et al., 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 each, **leaky-ReLU** activation
- Compare with gradient descent (AMSGrad)

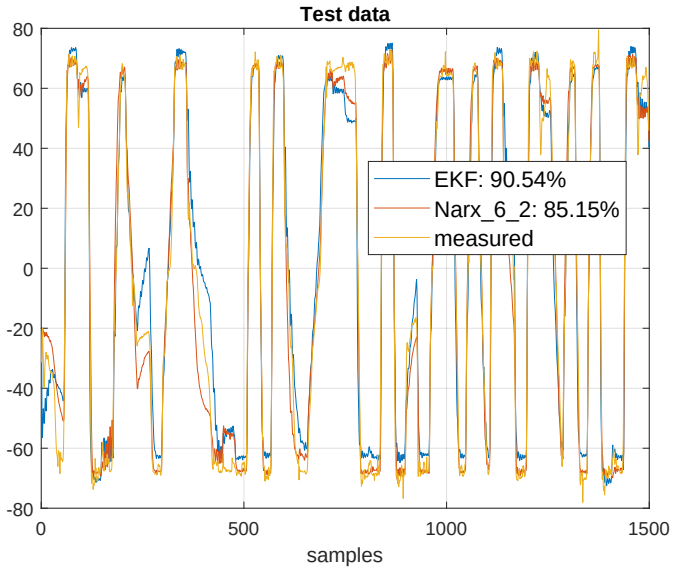


- Training time measured on MATLAB+CasADi implementation of EKF/AMSGrad

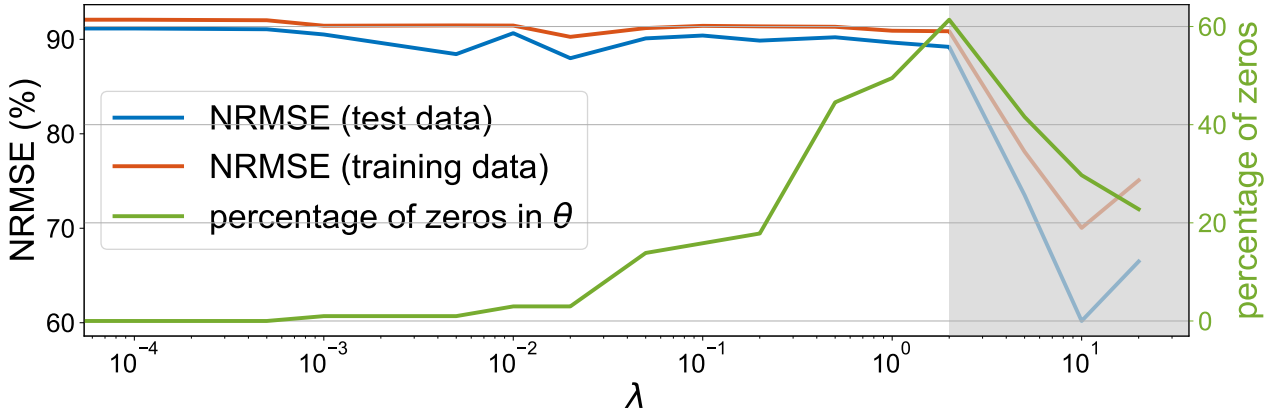
TRAINING RNNS BY EKF - EXAMPLES

- Compare NRMSE[1] wrt NNARX model (SYS-ID TBX):

  EKF = **91.97**, AMSGrad = **85.58**, NNARX(6,2) = **88.18** (**training**)
  EKF = **90.54**, AMSGrad = **80.95**, NNARX(6,2) = **85.15** (**test**)



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



[1] normalized root-mean-square error

TRAINING RNNS BY EKF - EXAMPLES

- Dataset: 2000 I/O data of linear system with **binary outputs**

$$x(k+1) = \begin{bmatrix} .8 & .2 & -.1 \\ 0 & .9 & .1 \\ .1 & -.1 & .7 \end{bmatrix} x(k) + \begin{bmatrix} -1 \\ .5 \\ 1 \end{bmatrix} u(k) + \xi(k) \qquad \mathrm{Var}[\xi_i(k)] = \sigma^2$$

$$y(k) = \begin{cases} \mathbf{1} & \text{if } \begin{bmatrix} -2 & 1.5 & 0.5 \end{bmatrix} x(k) - 2 + \zeta(k) \geq 0 \\ \mathbf{0} & \text{otherwise} \end{cases} \qquad \mathrm{Var}[\zeta(k)] = \sigma^2$$

- N=1000 data used for training, 1000 for testing the model

- Train **linear state-space model** with 3 states and **sigmoidal output** function

$$f_1^y(y) = 1/(1 + e^{-A_1^y [x'(k)\ u(k)]' - b_1^y})$$

σ	accuracy [%] training	test
0.000	99.20	98.90
0.001	99.30	98.90
0.010	99.20	98.70
0.100	96.50	97.00
0.200	93.00	93.80

- Training loss: (modified) **cross-entropy** loss

$$\ell_{\mathrm{CE}\epsilon}(y(k), \hat{y}) = \sum_{i=1}^{n_y} -y_i(k) \log(\epsilon + \hat{y}_i) - (1 - y_i(k)) \log(1 + \epsilon - \hat{y}_i)$$

TRAINING RNNS BY SEQUENTIAL LEAST-SQUARES

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

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

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

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

- Linearized model:

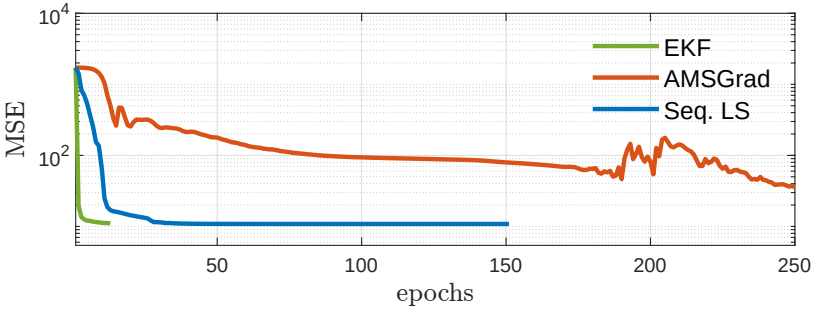
$$\begin{aligned} \Delta x_{k+1} &= (\nabla_x f_x)'\Delta x_k + (\nabla_{\theta_x} f_x)'\Delta\theta_x \\ \Delta y_k &= (\nabla_{x_k} f_y)'\Delta x_k + (\nabla_{\theta_y} f_y)'\Delta\theta_y \end{aligned}$$

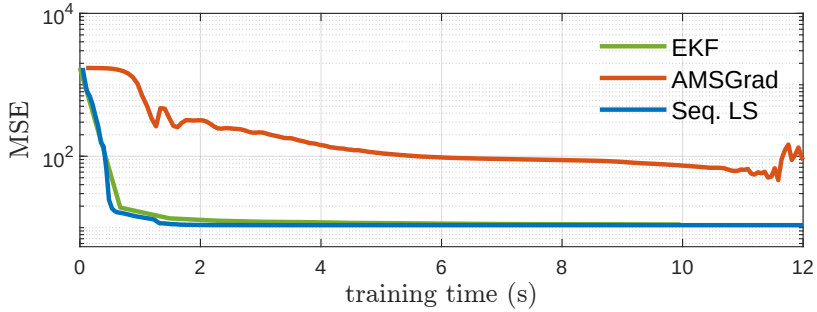
- **Idea**: take 2^{nd}-order expansions of the loss ℓ and regularization term r and use **sequential least-squares** + line search to minimize wrt x_0, θ_x, θ_y

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

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

- Fluid-damper example:





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

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

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

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

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

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

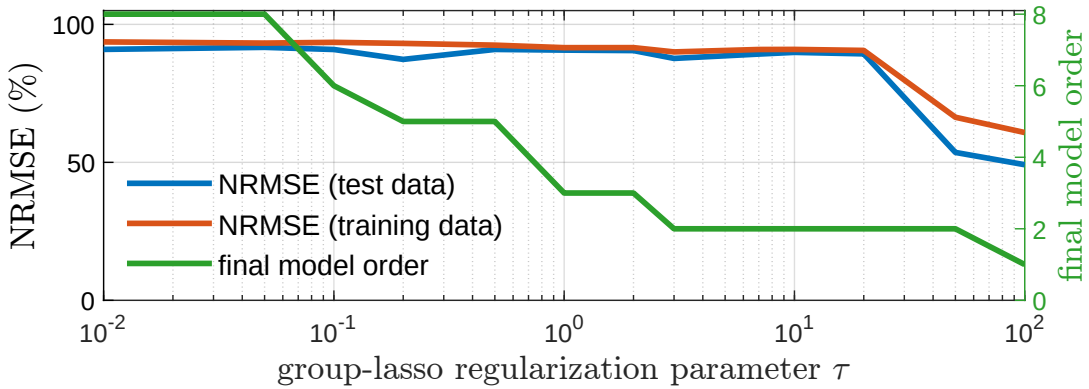
- ADMM + Seq. LS = **NAILS** algorithm (Nonconvex ADMM Iterations and Sequential LS)

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

$$\begin{bmatrix} \nu_x^{t+1} \\ \nu_y^{t+1} \end{bmatrix} = \text{prox}_{\frac{1}{\rho}g}(\theta_x^{t+1} + w_x^t, \theta_y^{t+1} + w_y^t)$$ 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: **group-Lasso regularization** $g(\nu_i^g) = \tau \sum_{i=1}^{n_x} \|\nu_i^g\|_2$ to zero entire rows and columns and **reduce state-dimension** automatically



TRAINING RNNS BY SEQUENTIAL LS AND ADMM

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

- Fluid-damper example: **quantization** of θ_x, θ_y for simplifying model arithmetic +ReLU activation function

$$g(\theta_i) = \begin{cases} 0 & \text{if } \theta_i \in \mathcal{Q} \\ +\infty & \text{otherwise} \end{cases}$$

$\mathcal{Q}$ = multiples of 0.1 between -0.5 and 0.5

  - NRMSE = **83.10** (training), **80.51** (test)
  - NRMSE = **8.83** (training), **2.69** (test) ← **no ADMM, just quantize after training**
  - Training time: ≈ 5 s

- **Note**: no convergence to a global minimum is guaranteed

- **NAILS** = very flexible & efficient learning algorithm for **control-oriented RNNs**

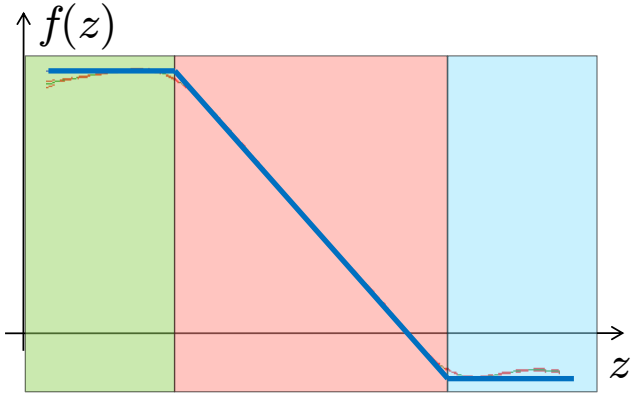
LEARNING HYBRID PREDICTION MODELS

LEARNING HYBRID MODELS

- **Switching dynamics** are better captured by **piecewise affine (PWA)** models and handled by **hybrid MPC** techniques

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

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



- **PWA regression**: learn **both** the $\{F_i,\ g_i\}$ **and** the partition $\{H_i,\ K_i\}$

(Ferrari-Trecate, Muselli, Liberati, Morari, 2003) (Roll, Bemporad, Ljung, 2004) (Juloski, Wieland, Heemels, 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, 2021, arXiv)

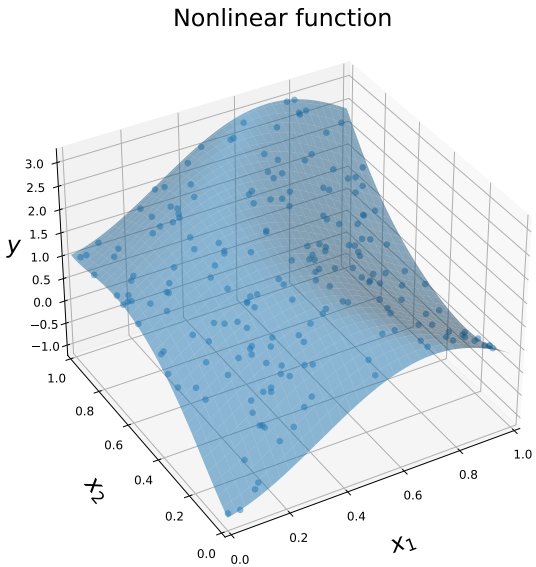
- New **Piecewise Affine Regression and Classification** (**PARC**) algorithm
- Training dataset:
  - **feature vector** $z \in \mathbb{R}^n$ (categorical features **one-hot encoded** in $\{0,1\}$)
  - **target vector** $v_c \in \mathbb{R}^{m_c}$ (numeric), $v_{di} \in \{w_{di}^1, \ldots, w_{di}^{m_i}\}$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

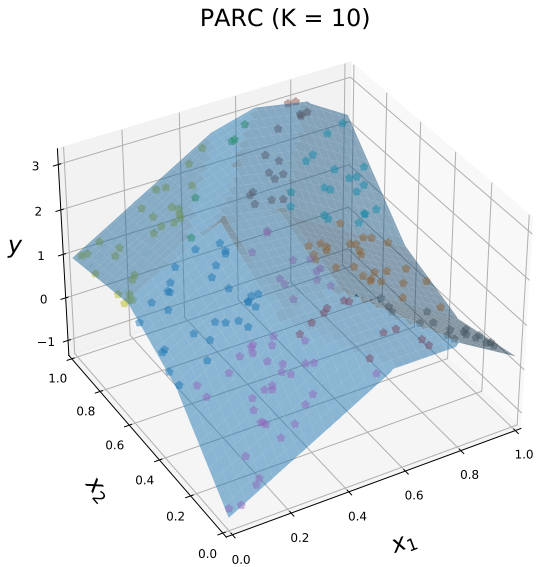
$$\Phi(z) = \omega^{j(z)} z + \gamma^{j(z)}, \qquad j(z) = \min\left\{\arg\max_{j=1,\ldots,K}\{\omega^j z + \gamma^j\}\right\}$$

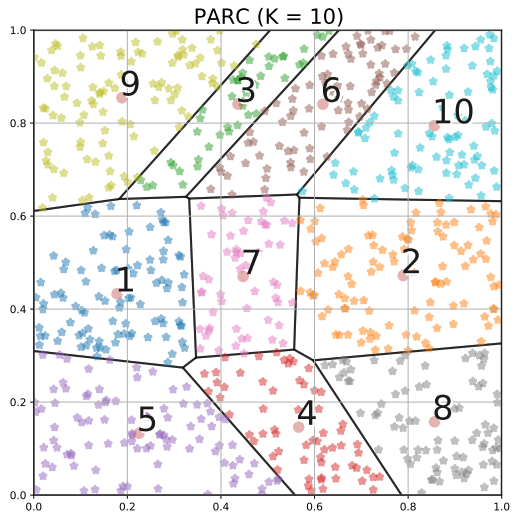
- Data reassigned to clusters based on weighted fit/PWL separation criterion
- PARC is a **block-coordinate descent** algorithm $\Rightarrow$ (local) convergence ensured

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

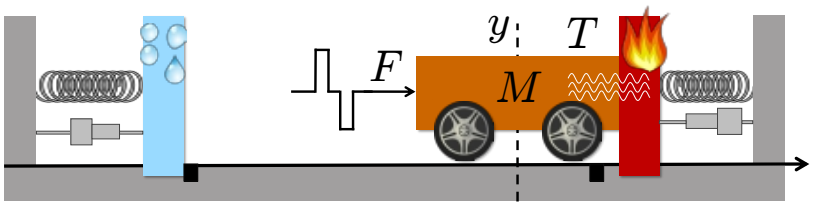






- Code download: **http://cse.lab.imtlucca.it/~bemporad/parc/**

DATA-DRIVEN HYBRID-MPC EXAMPLE



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

  Spring and viscous forces are **nonlinear**.

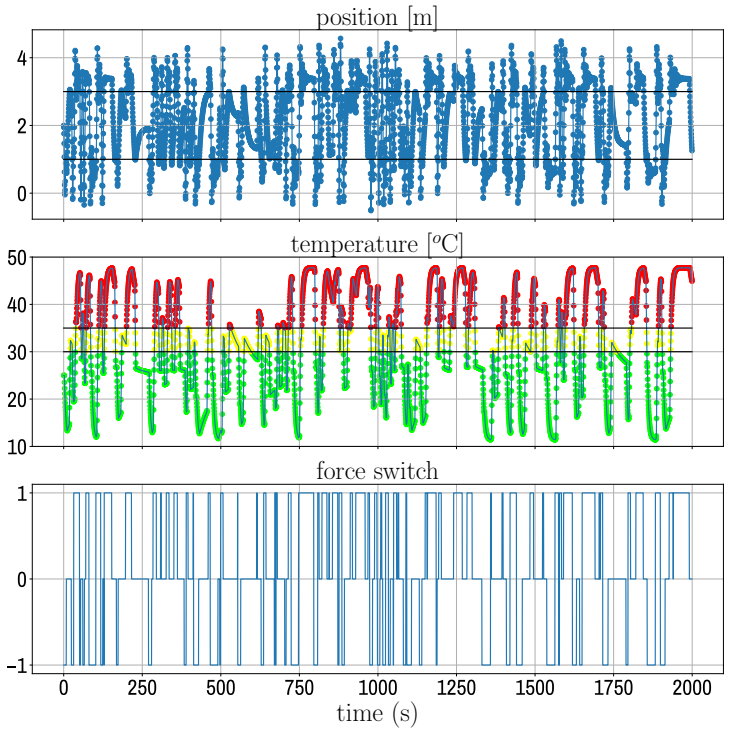
- Categorical input $F \in \{-\bar{F}, 0, \bar{F}\}$ and categorical output $c \in \{green, yellow, red\}$

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

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

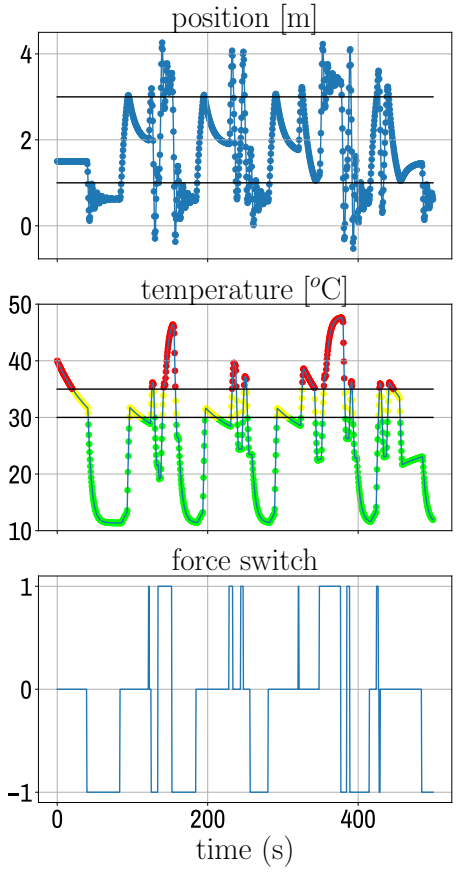
- Target vector $v_k = [y_{k+1}, \dot{y}_{k+1}, T_{k+1}, c_k]$

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

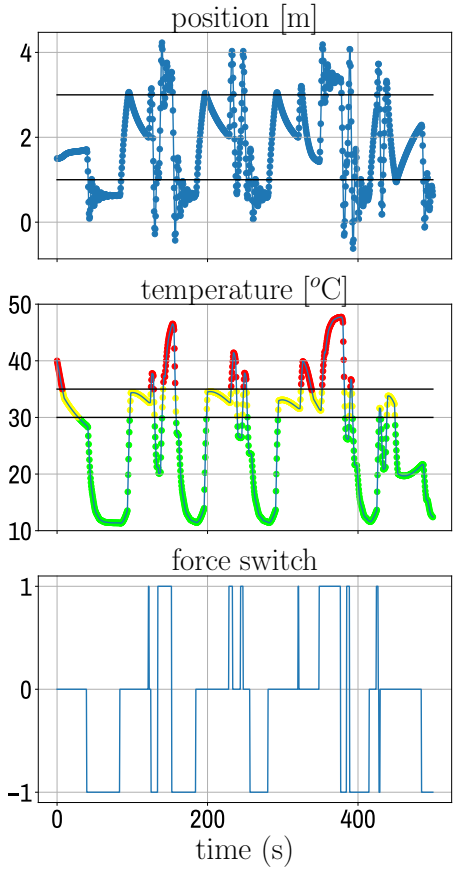


DATA-DRIVEN HYBRID-MPC EXAMPLE

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



continuous-time system



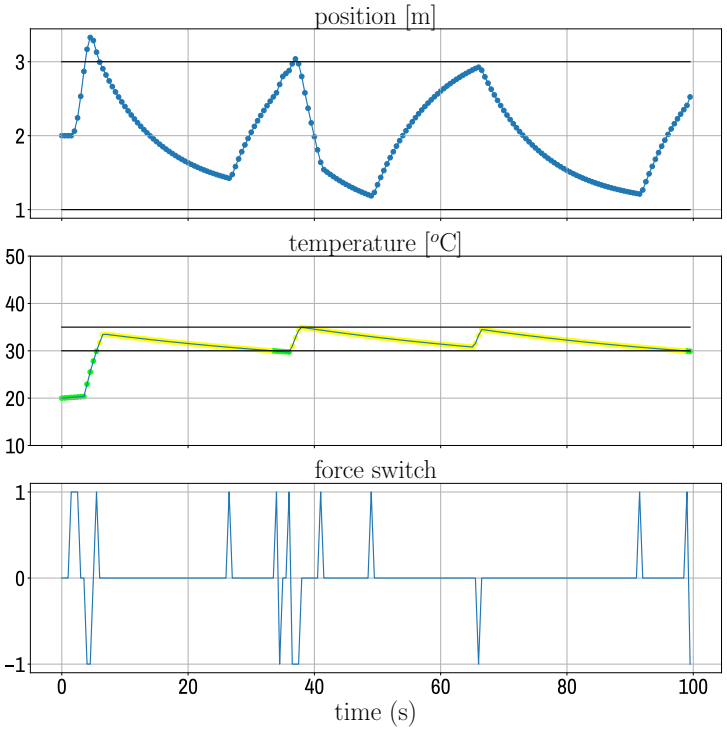
discrete-time PWA model

- Model fit is good enough for MPC design purposes

DATA-DRIVEN HYBRID-MPC EXAMPLE

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

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



- Problem can be cast to MILP.
  Solution time: 0.15-0.29 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: 99.9%). CPU time = 52÷67 μs. Closed-loop trajectories very similar.

LEARNING OPTIMAL MPC CALIBRATION

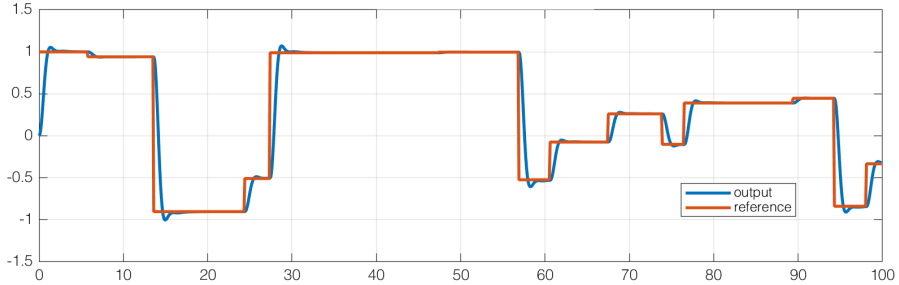
MPC CALIBRATION PROBLEM

- The design depends on a vector x of **MPC parameters**
- MPC parameters are intuitive to set (e.g., weights)
- Still, can we auto-calibrate them ?

- 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)
  - ...
- **New method**: **radial basis function** surrogates + **inverse distance weighting** (**GLIS**) (Bemporad, 2020)

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

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:

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

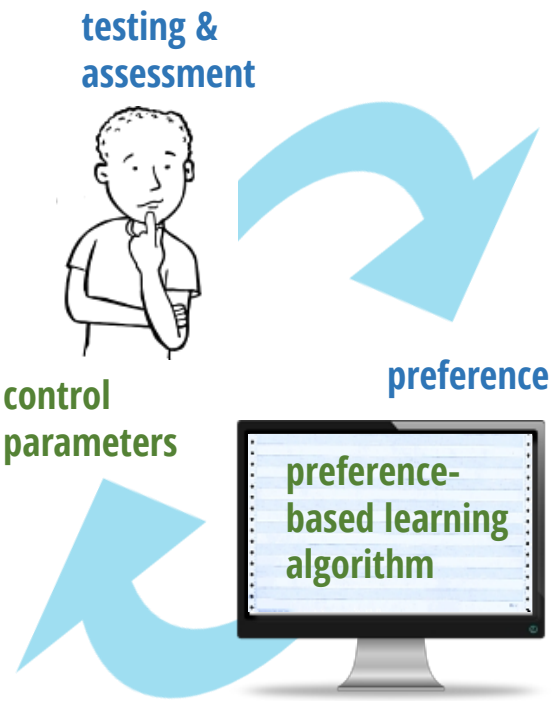
- We want to find a global optimum $x^\star$ (="better" than any other x)

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

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

SEMI-AUTOMATIC TUNING 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**

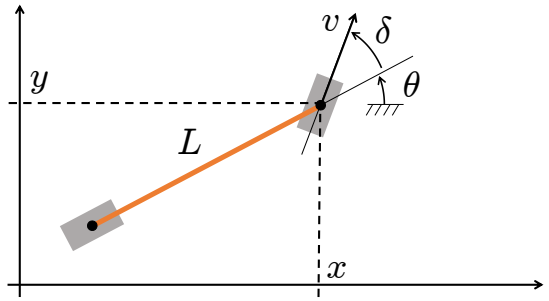


PREFERENCE-BASED TUNING: MPC EXAMPLE

(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:

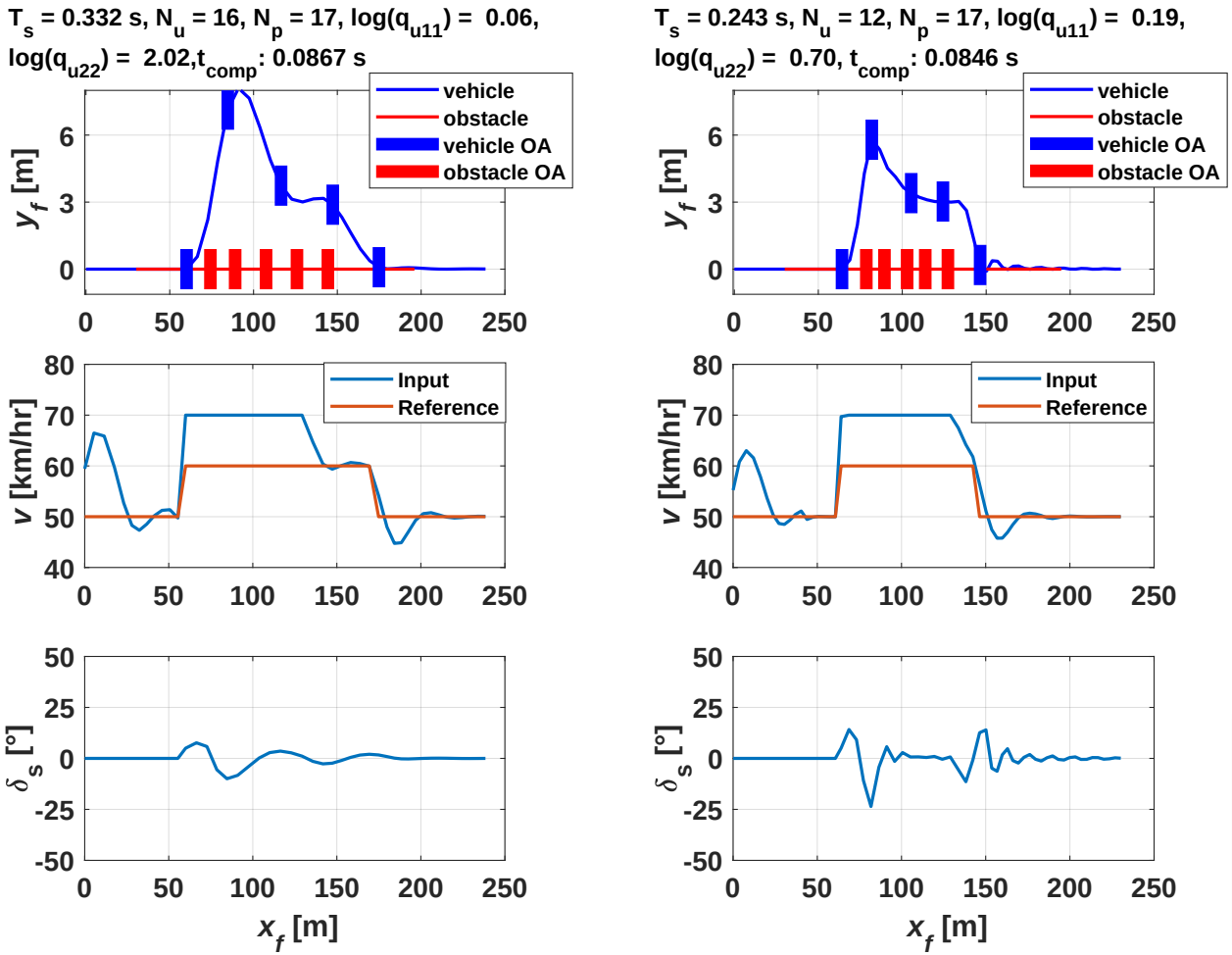
  "optimal obstacle avoidance", "pleasant drive", "keep CPU time small", …

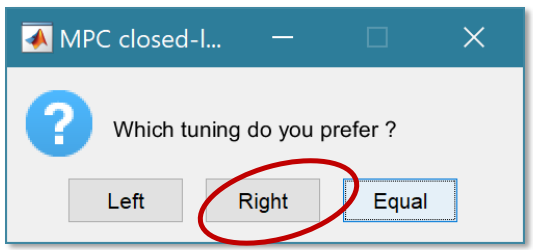
  → **not easy to quantify in a single function**

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

PREFERENCE-BASED TUNING: MPC EXAMPLE

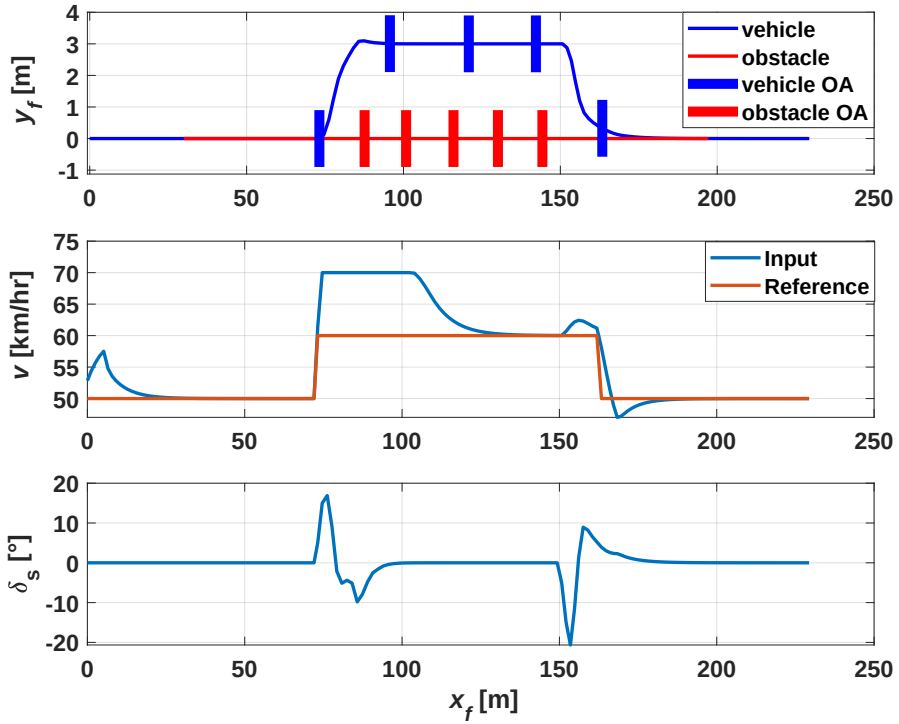
- Preference query window:





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)

CONCLUSIONS

- **Learning-based MPC** is a formidable combination for advanced control:
  - **MPC** / on-line optimization is an extremely powerful control methodology
  - **ML** extremely useful to get **control-oriented models** and **control laws** from **data**
- Ignoring **ML** tools would be a mistake (a lot to "learn" from machine learning)
- **ML** cannot replace control engineering:
  - **Black-box** modeling can be a failure. Better use **gray-box** models when possible
  - Approximating the control law can be a failure. Don't abandon on-line optimization
  - Pure AI-based **reinforcement learning** methods can be also a failure
- A wide spectrum of research opportunities and new practices is open !