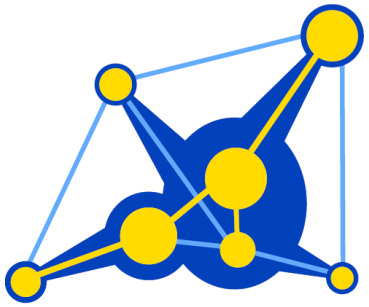
Algorithms for Nonlinear MPC of Large Scale Systems

Moritz Diehl

Optimization in Engineering Center (OPTEC)
& Electrical Engineering Department (ESAT),
K.U. Leuven, Belgium

Industry Workshop on Hierarchical
and Distributed Model Predictive Control

Leuven, June 24, 2011



OPTEC - Optimization in Engineering Center

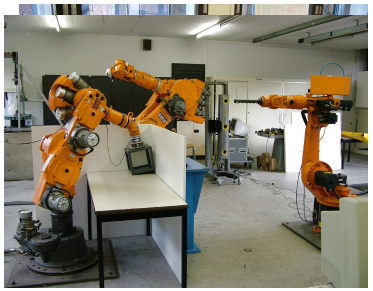
Center of Excellence of K.U. Leuven, from 2005-2010, 2010-2017

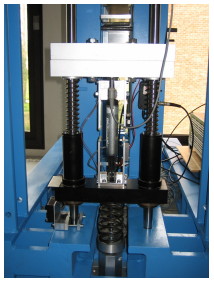
About 20 professors, 10 postdocs, and 40 PhD students involved in OPTEC research

Scientists in 5 divisions:

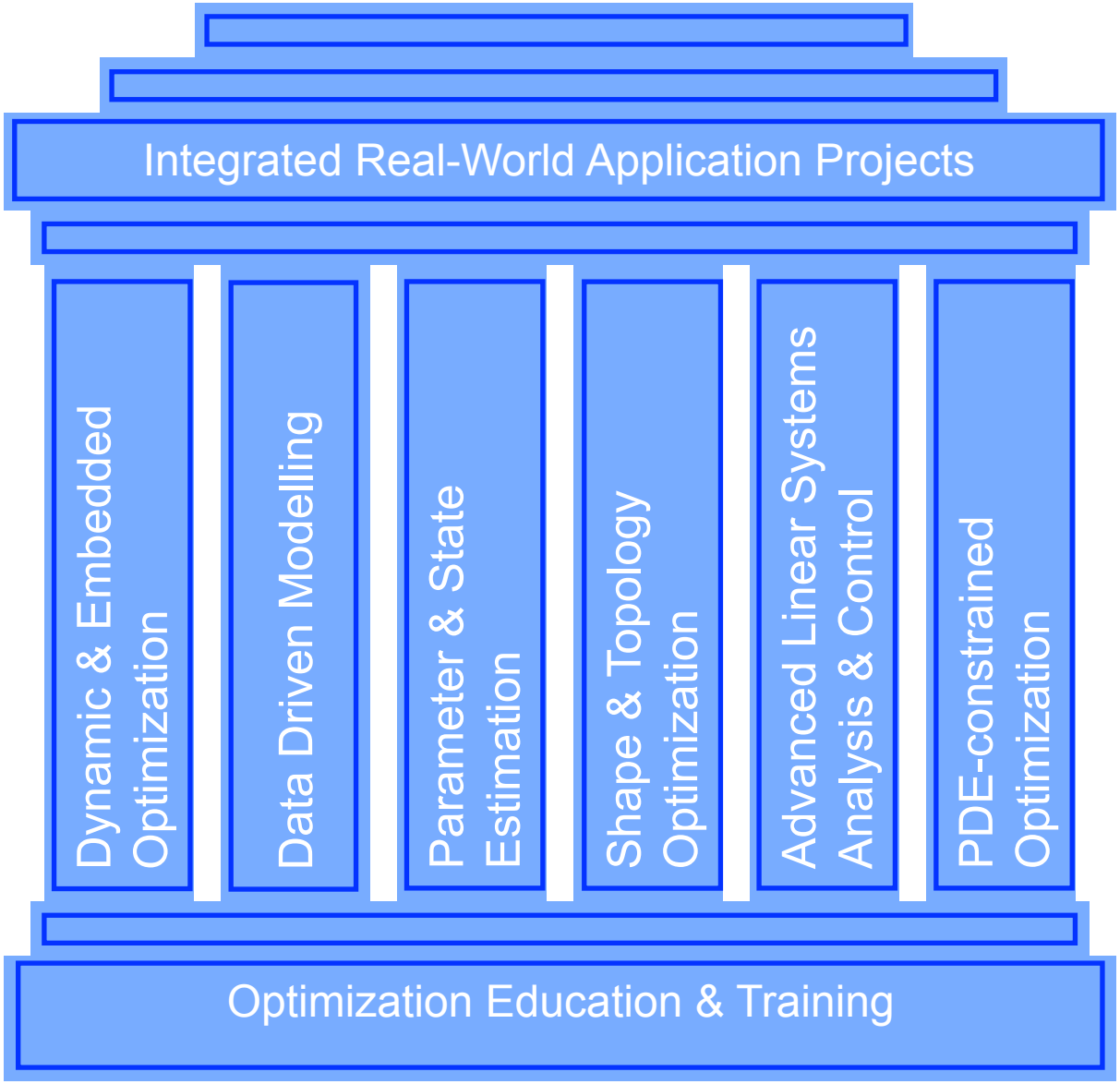
- Electrical Engineering
- Mechanical Engineering
- Chemical Engineering
- Computer Science
- Civil Engineering

Many real world applications at OPTEC...

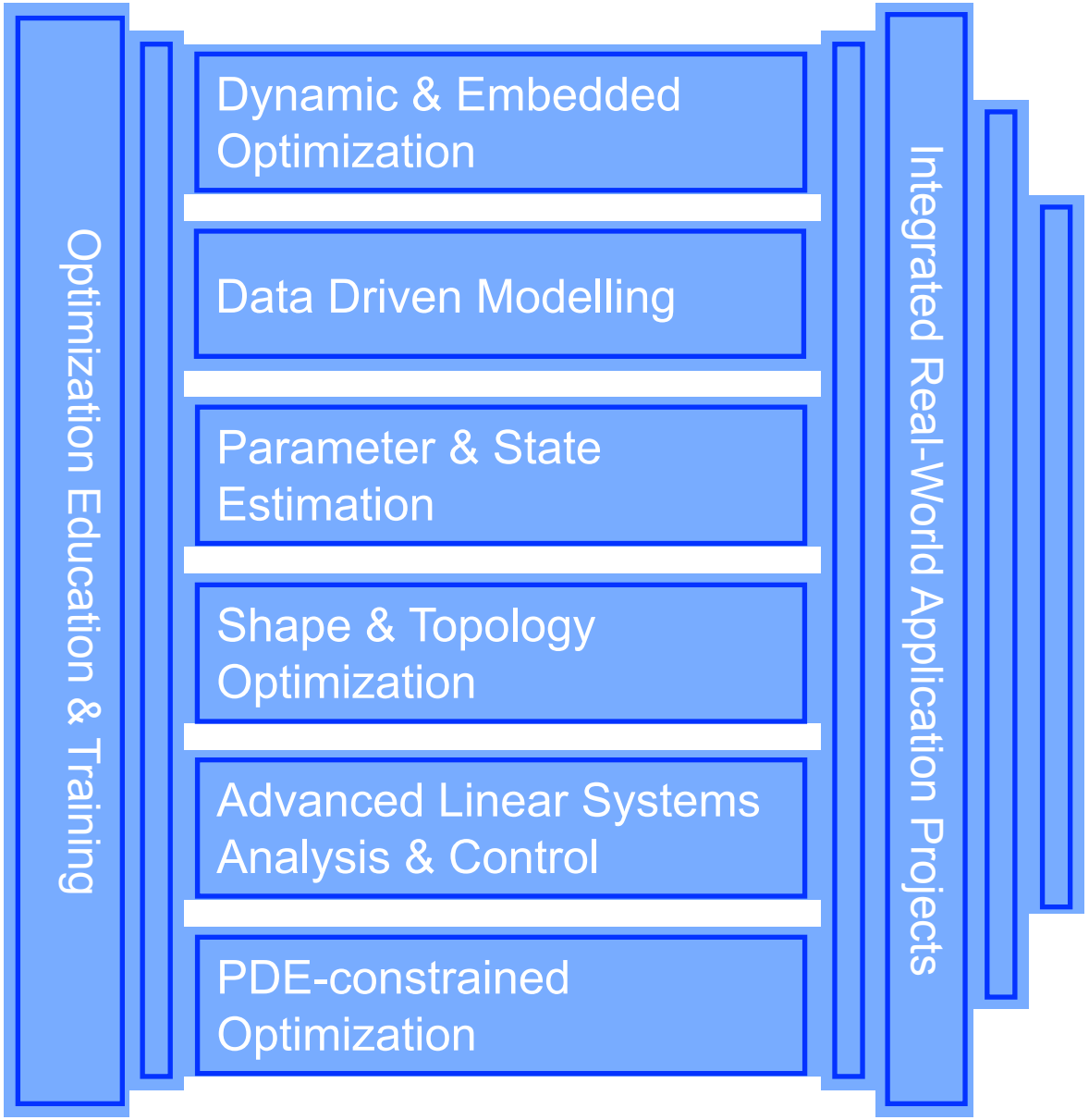




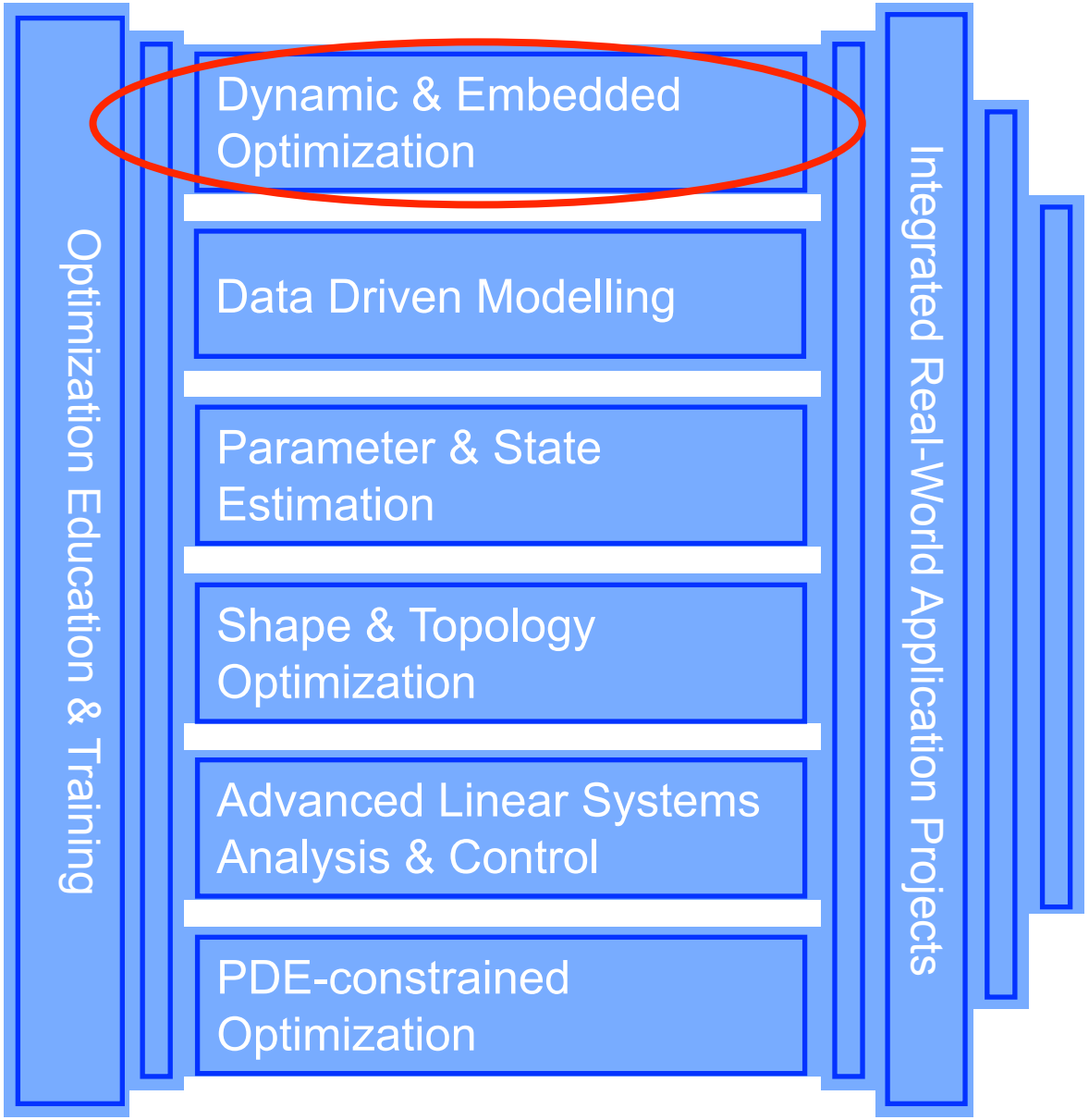
OPTEC: 70 people in six methodological working groups



OPTEC: 70 people in six methodological working groups



OPTEC: 70 people in six methodological working groups



Overview

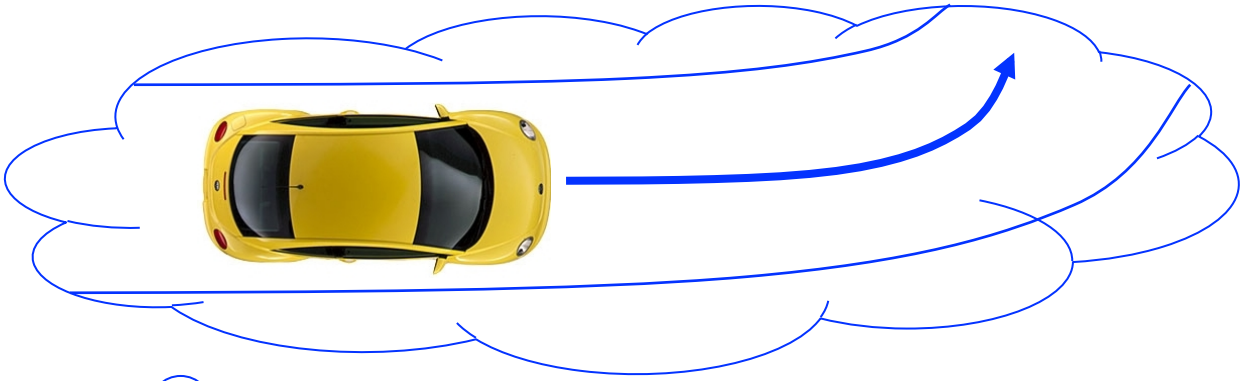
- Linear MPC for Mechatronic System and qpOASES
- Nonlinear MPC for Distillation Control
- Nonlinear MPC with ACADO Toolkit
- Modelica and Automatic Derivative Generation using CasADI
- Distributed MPC (outlook → C. Savorgnan)

Linear MPC in Mechatronics


with **Lieboud Vanden Broeck, Hans Joachim Ferreau,** Jan Swevers

Model Predictive Control (MPC)

Always look a bit into the future.



Brain predicts and optimizes:
e.g. slow down **before** curve

Linear MPC = parametric QP

For

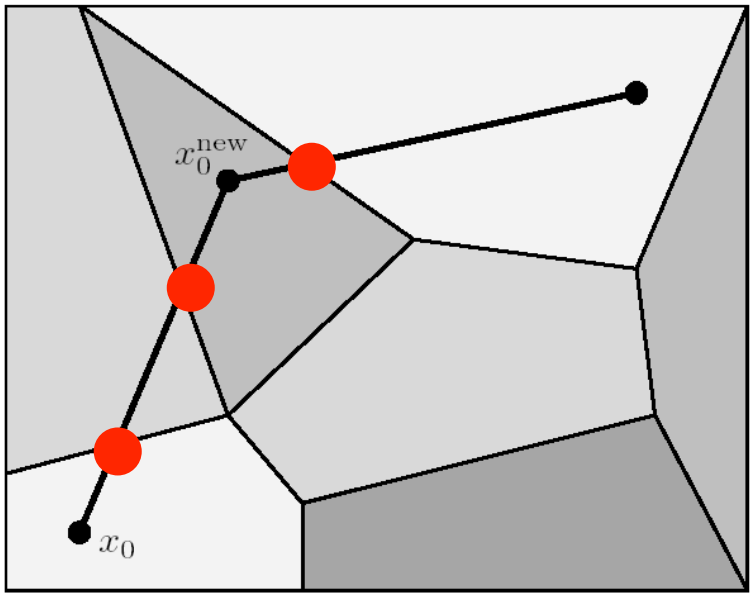
- linear dynamic system $x_{k+1} = Ax_k + Bu_k\,,$
- linear constraints
- quadratic cost

Only *parametric quadratic program (p-QP)* needs to be solved:

$$
\begin{aligned}
\min_{\substack{u_0,\dots,u_{N-1}\\ x_1,\dots,x_N}} \quad & x_N^T P x_N + \sum_{i=0}^{N-1} \left(x_i^T Q x_i + u_i^T R u_i\right) \\
\text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k\,, \\
& (x_0 \text{ given})\,, \\
& \underline{c} \le Cx_k \le \overline{c}\,, \\
& \underline{d} \le Du_k \le \overline{d}\,, \\
& c_T \le C_T x_N\,,
\end{aligned}
$$

Online Active Set Strategy

Solve p-QP via „Online Active Set Strategy“:



- go on straight line in parameter space from old to new problem data
- **solve each QP on path exactly (keep primal-dual feasibility)**
- Update matrix factorization at boundaries of critical regions
- Up to 10 x faster than standard QP

qpOASES: open source C++ code by Hans Joachim Ferreau

qpOASES – Online Active Set Strategy

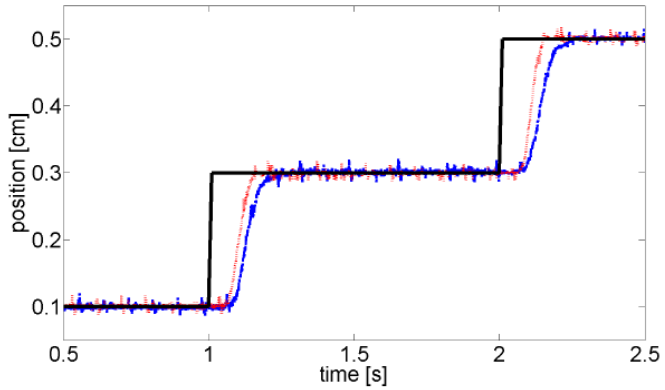
- qpOASES is an **object-oriented C++ implementation** of the online active set strategy with dense linear algebra
  (see www.qpOASES.org, version 3.0beta to be released)

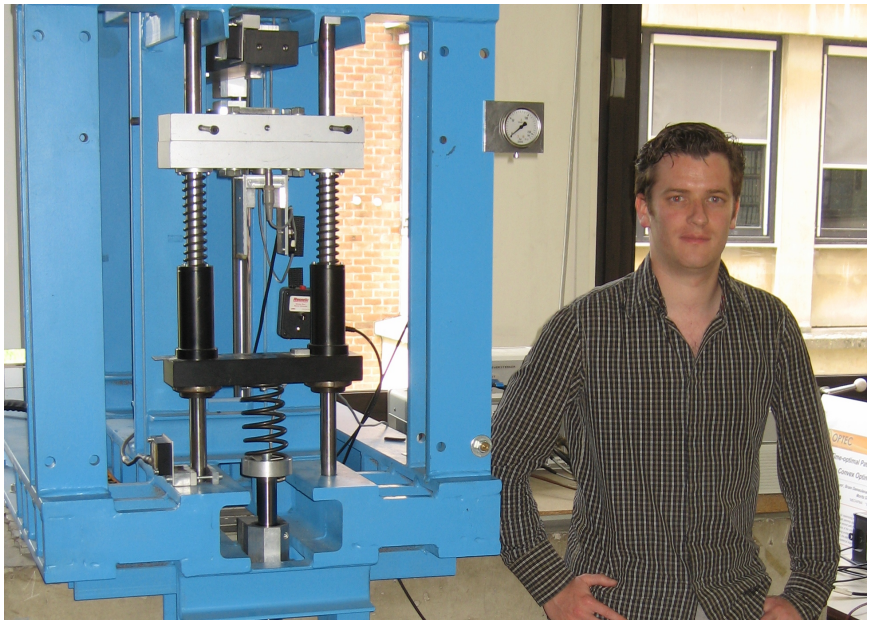
- Distributed as **open-source software** under the GNU LGPL

- **Self-contained code**: no additional software packages required (but BLAS/LAPACK can be linked)

- **Interfaces** to several third-party software packages:
  - Matlab, Octave, Scilab
  - Simulink (dSPACE, xPC Target)
  - ACADO Toolkit
  - YALMIP (under development)

Time Optimal MPC: a 100 Hz Application

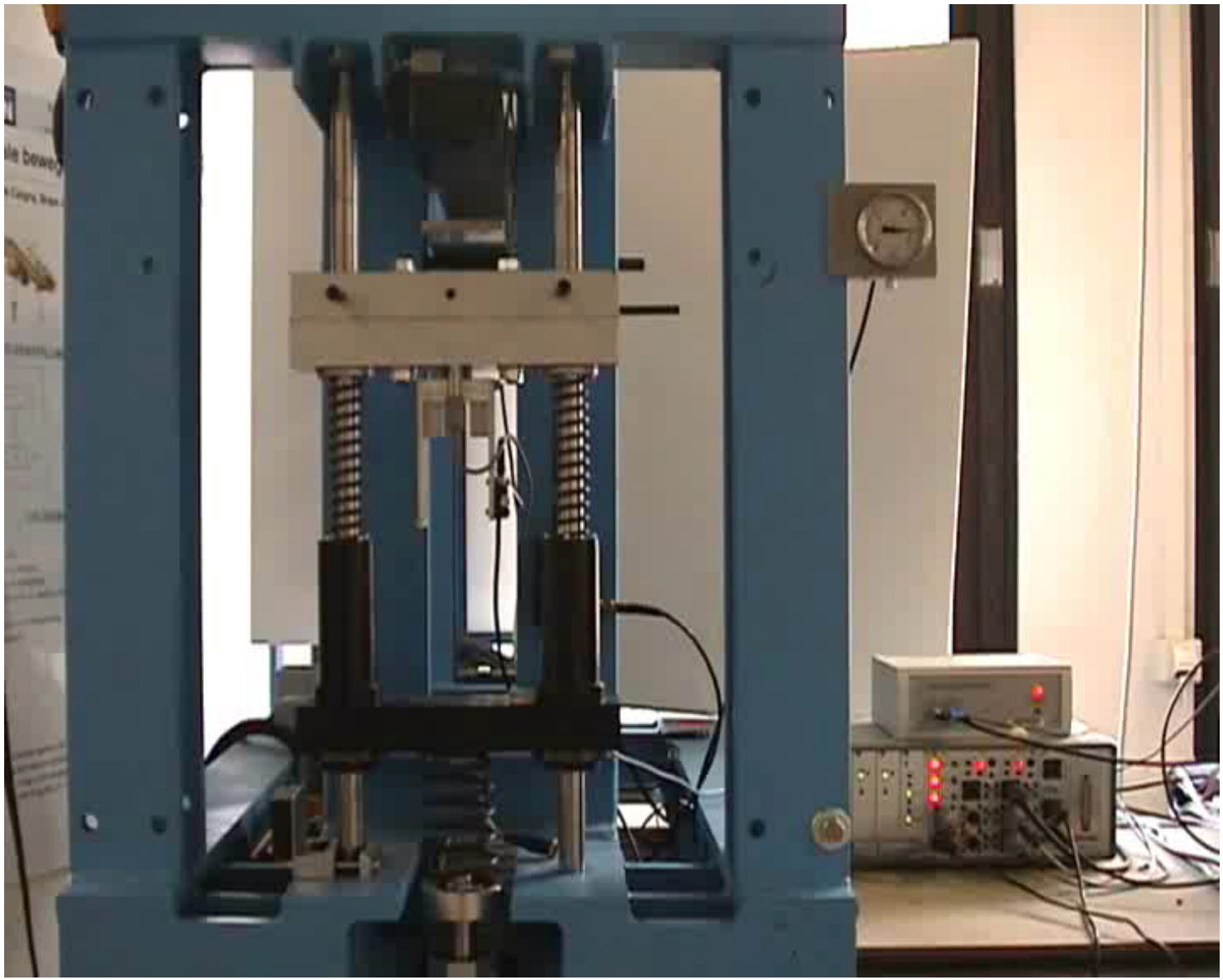




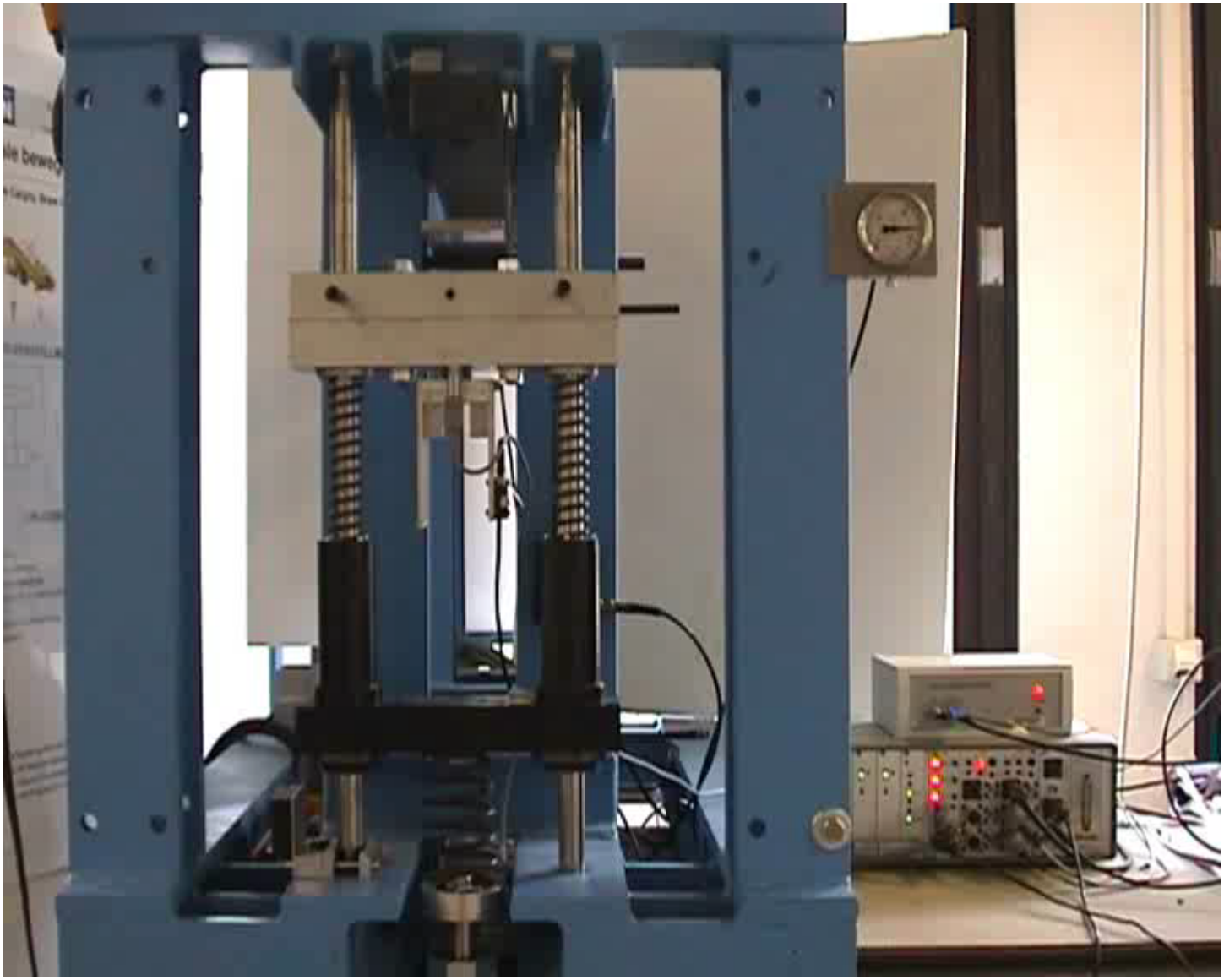
- Quarter car: oscillating spring damper system
- MPC Aim: settle at any new setpoint in ***in minimal time***
- Two level algorithm: MIQP
  - 6 online data
  - 40 variables + one integer
  - 242 constraints (in-&output)
- use qpOASES on dSPACE
- **CPU time: <10 ms**

Lieboud Van den Broeck in front of quarter car experiment

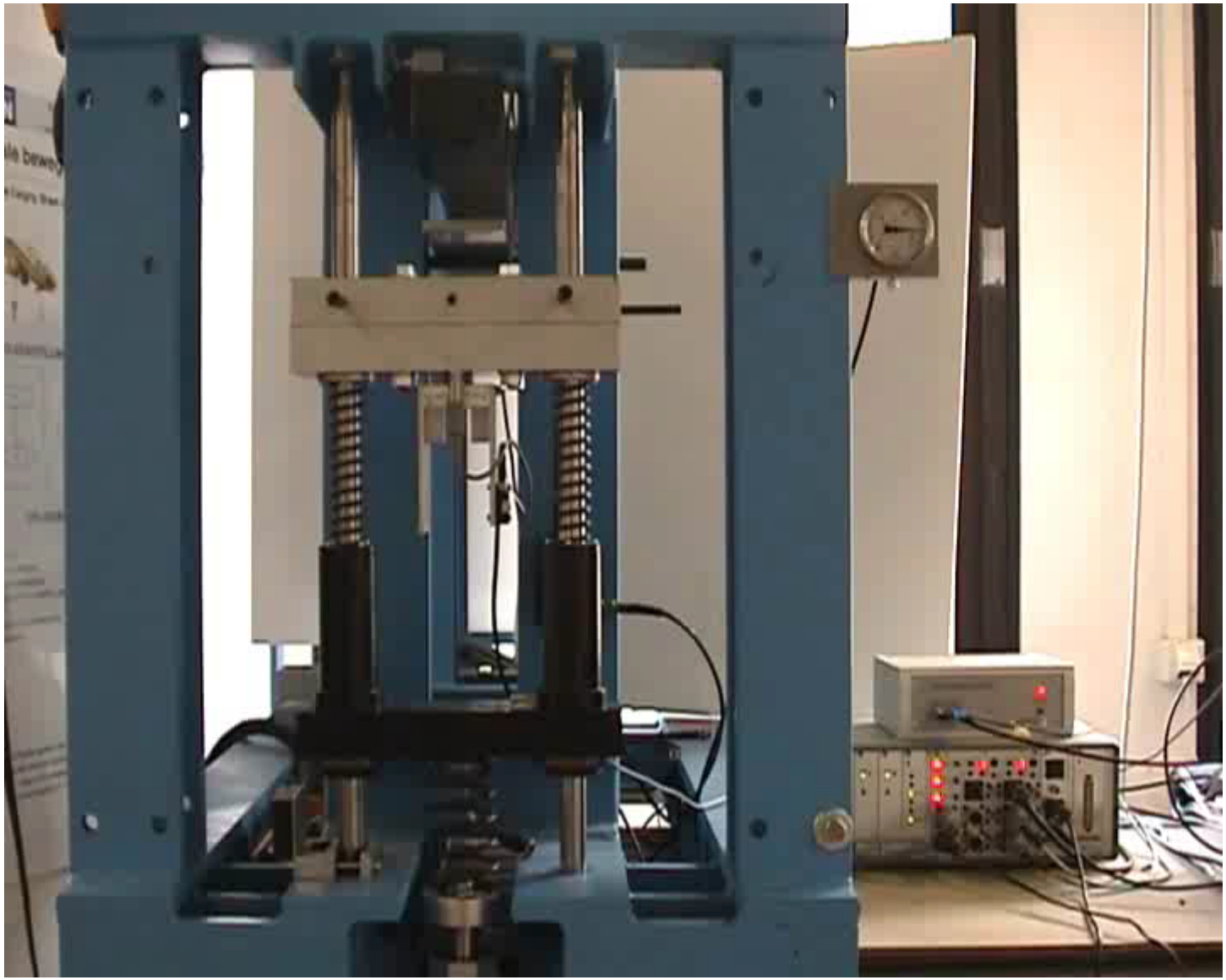
Setpoint change without control: oscillations



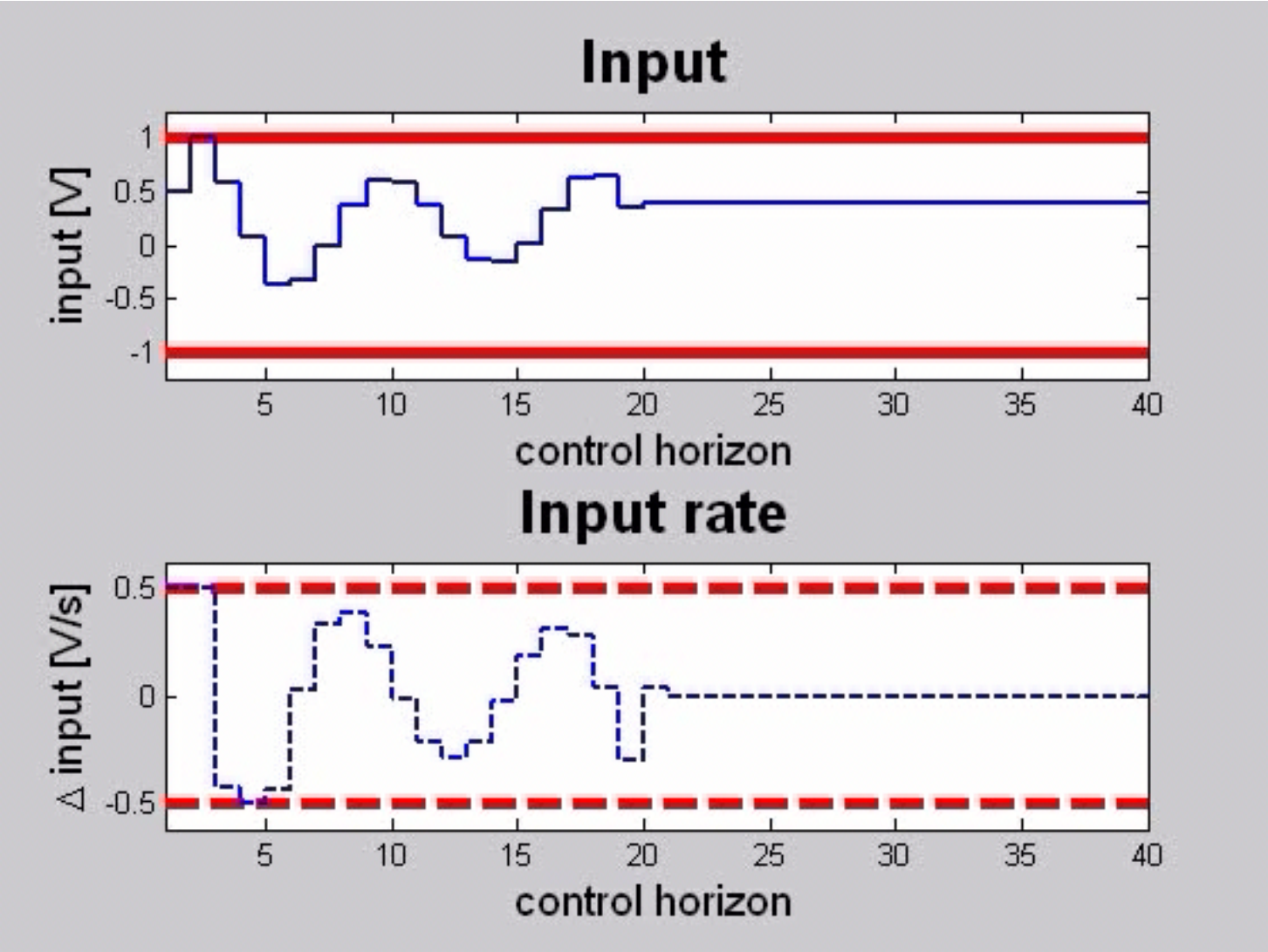
With LQR control: inequalities violated



With Time Optimal MPC



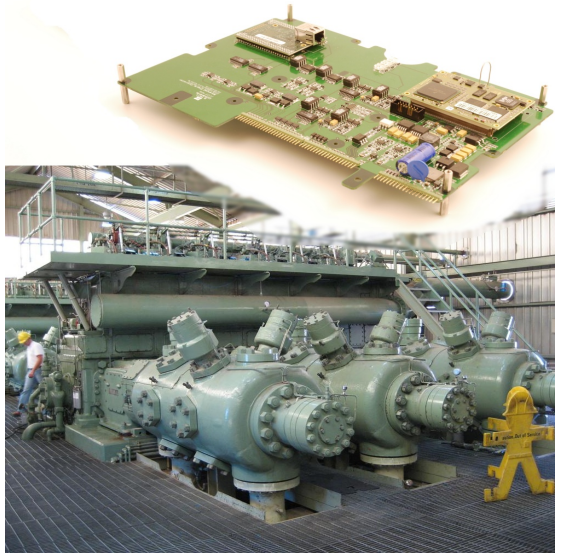
Time Optimal MPC: qpOASES Optimizer Contents



qpOASES running on Industrial Control Hardware (20 ms)

Project manager (Dec. 2008): “…we had NO problem at all with the qpOASES code. Your Software has throughout the whole project shown reliable and robust performance.”

qpOASES – Online Active Set Strategy



- Reliable code used in many academic and **industrial real-world applications**:
  - Integral gas engines (Hoerbiger)
  - Diesel engine testbench
  - Machine tools
  - Walking robots
  - MPC for Process Industry (INCA Suite of IPCOS)
- Dense linear algebra, but sparsity can be partly exploited
- Solves dense convex QPs with up to 1000 variables and a few thousand constraints (in a couple of seconds)
- Small-scale QPs are typically solved within a few milli/microseconds

Nonlinear Model Predictive Control of a Distillation Column in Stuttgart

with Frank Allgower, Rolf Findeisen, Hans Georg Bock, Ilknur Uslu, Stefan Schwarzkopf, Zoltan Nagy

First Principle Dynamic System Models

$$\dot{n}_{N+1} = V_N - D - L_{N+1}.$$

$$0 = \dot{n}_\ell^v = V^m(X_\ell, T_\ell)\dot{n}_\ell + \frac{\partial V^m}{\partial(X,T)}(\dot{X}_\ell, \dot{T}_\ell)^T\, n_\ell,$$

$$L_{\text{vol}} = V^m(X_{N+1}, T_C)L_{N+1},$$

$$\dot{X}_0 n_0 + X_0 \dot{n}_0 = -V_0 Y_0 + L_1 X_1 - B X_0,$$

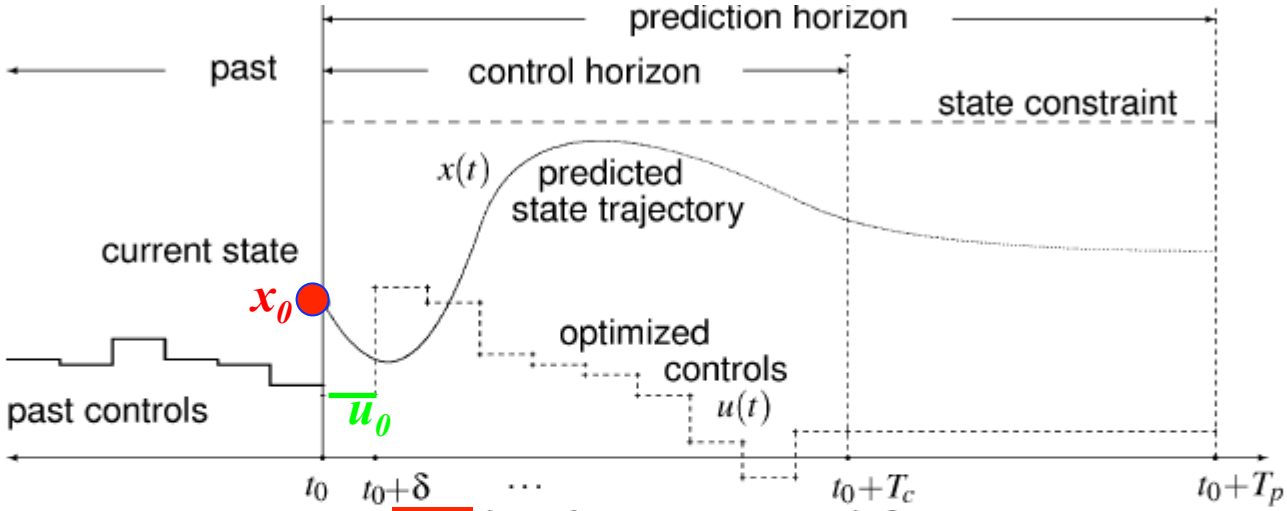
$$\dot{X}_\ell n_\ell + X_\ell \dot{n}_\ell = V_{\ell-1}Y_{\ell-1} - V_\ell Y_\ell + L_{\ell+1}X_{\ell+1} - L_\ell X_\ell$$
$$\ell = 1, 2, \ldots, N_F - 1, N_F + 1, \ldots, N$$

$$\dot{X}_{N_F} n_{N_F} + X_{N_F}\dot{n}_{N_F} = V_{N_F-1}Y_{N_F-1} - V_{N_F}Y_{N_F} + L_{N_F+1}X_{N_F+1} - L_{N_F}X_{N_F} + FX_F,$$

- Nonlinear differential algebraic equations (DAE)
- often in modeling languages like gPROMS, SIMULINK, Modelica
- Hundreds to thousands of states

Can we use these models directly for optimization and feedback control?

Computations in Nonlinear MPC



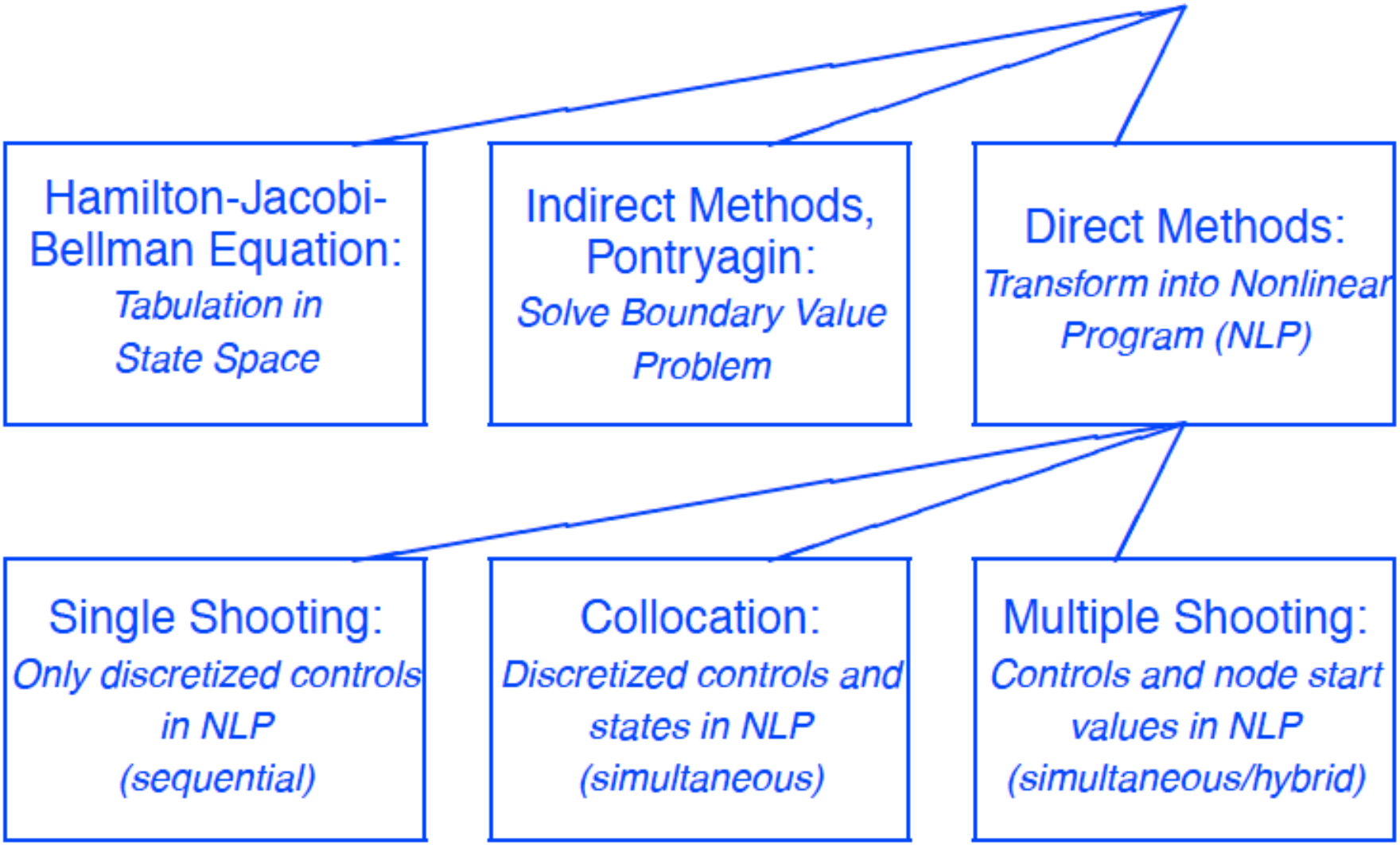
1. Estimate current system state x_0 (and parameters) from measurements.
2. Solve *in real-time* an optimal control problem:

$$\min_{x,z,u} \int_{t_0}^{t_0+T_p} L(x,z,u)dt + E\left(x(t_0+T_p)\right) \; s.t. \begin{cases} x(t_0) - x_0 = 0, \\ \dot{x} - f(x,z,u) = 0, \; t \in [t_0, t_0+T_p] \\ g(x,z,u) = 0, \; t \in [t_0, t_0+T_p] \\ h(x,z,u) \geq 0, \; t \in [t_0, t_0+T_p] \\ r\left(x(t_0+T_p)\right) \geq 0. \end{cases}$$

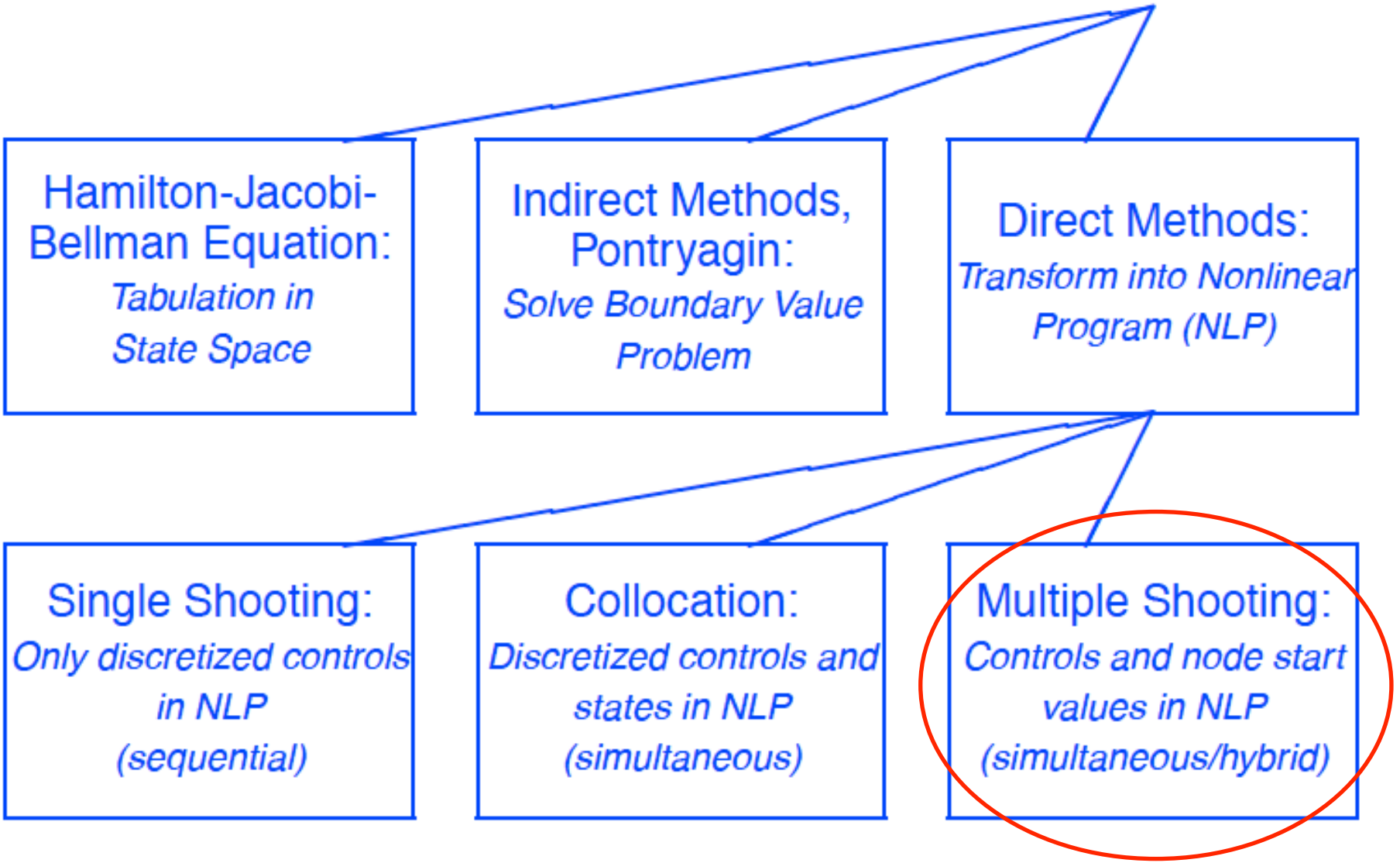
3. Implement first control u_0 for time δ at real plant. Set $t_0 = t_0 + \delta$ and go to 1.

Main challenge for NMPC: fast and reliable real-time optimization

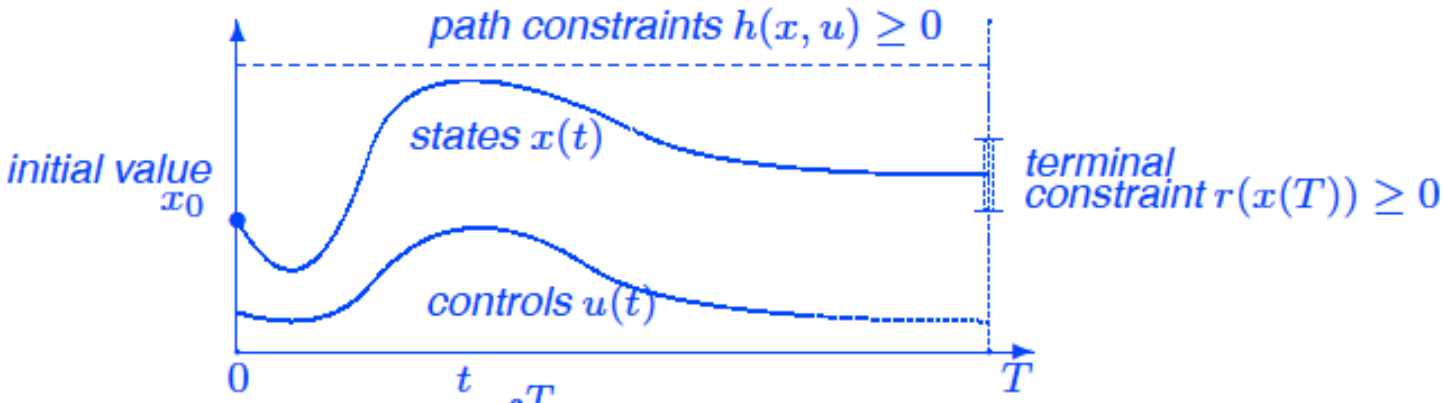
Optimal Control Family Tree



Optimal Control Family Tree



Simplified Optimal Control Problem in ODE



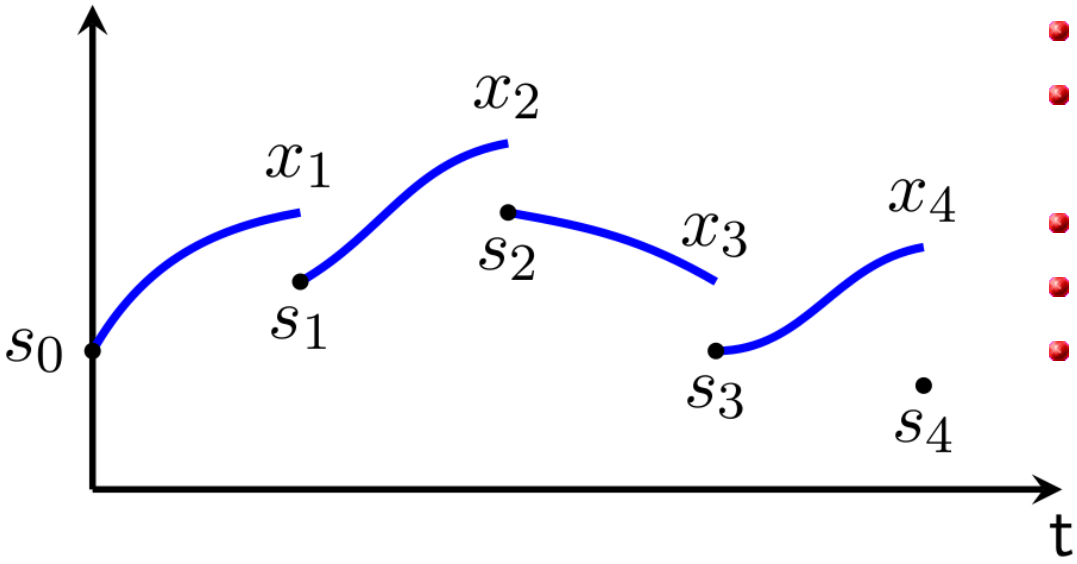
$$\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad \int_0^T L(x(t),u(t))\,dt \;+\; E\,(x(T))$$

subject to

$$\begin{aligned}
x(0) - x_0 &= 0, && && \text{(fixed initial value)}\\
\dot{x}(t) - f(x(t),u(t)) &= 0, && t \in [0,T], && \text{(ODE model)}\\
h(x(t),u(t)) &\geq 0, && t \in [0,T], && \text{(path constraints)}\\
r\,(x(T)) &\geq 0 && && \text{(terminal constraints).}
\end{aligned}$$

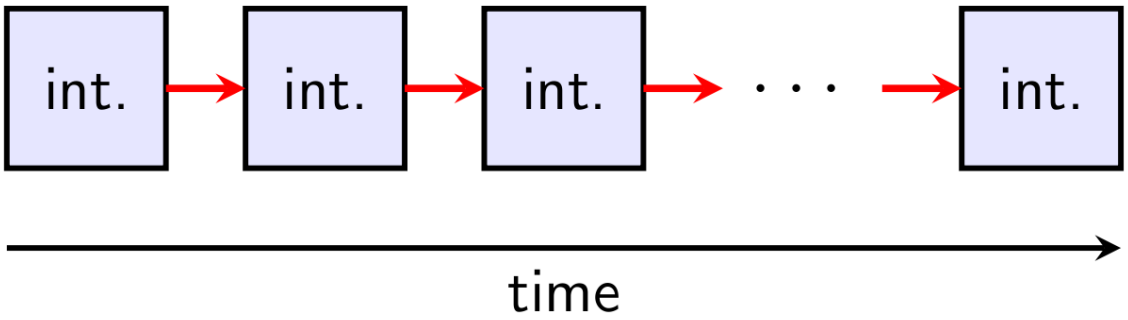
Direct Multiple Shooting [Bock & Plitt 1984]

Simulate system on intervals, solve nonlinear program (NLP)

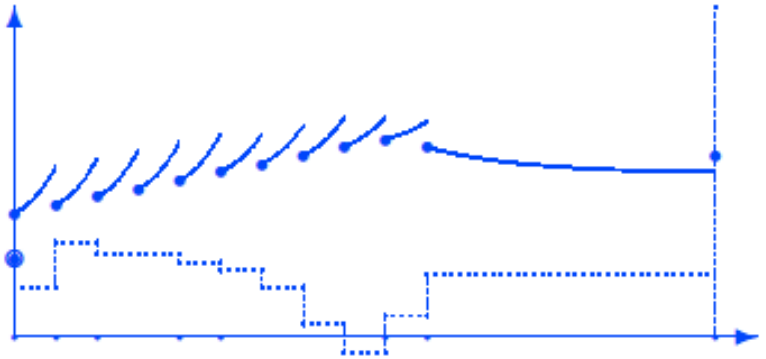


- gap between shooting intervals
- constraint forces gaps to zero

- freedom for initialization
- suitable for unstable systems
- parallelizable



NLP in Direct Multiple Shooting



$$\underset{s,\,q}{\text{minimize}} \quad \sum_{i=0}^{N-1} l_i(s_i, q_i) \;+\; E\left(s_N\right)$$

subject to

$$\begin{aligned}
s_0 - x_0 &= 0, && && \text{(initial value)}\\
s_{i+1} - x_i(t_{i+1}; s_i, q_i) &= 0, && i = 0, \ldots, N-1, && \text{(continuity)}\\
h(s_i, q_i) &\geq 0, && i = 0, \ldots, N, && \text{(discretized path constraints)}\\
r\left(s_N\right) &\geq 0. && && \text{(terminal constraints)}
\end{aligned}$$

Example: Distillation Column (ISR, Stuttgart)

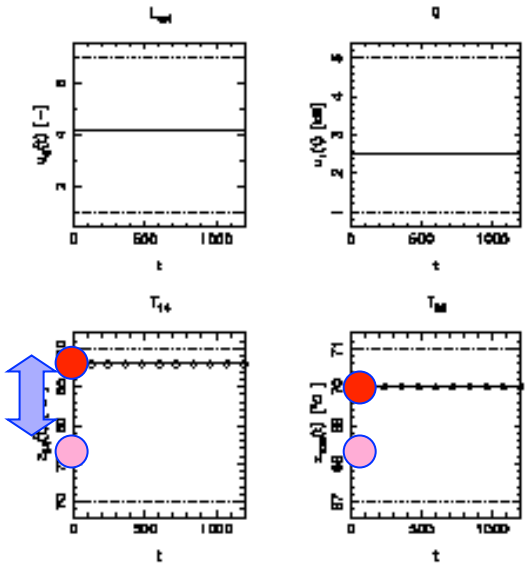
- Aim: to ensure product purity, keep two temperatures (T_{14}, T_{28}) constant despite disturbances
- least squares objective:

$$\min \int_{t_0}^{t_0+T_p} \left\| \begin{array}{c} T_{14}(t) - T_{14}^{\text{ref}} \\ T_{28}(t) - T_{28}^{\text{ref}} \end{array} \right\|_2^2 dt$$

- control horizon 10 min
- prediction horizon 10 h
- stiff DAE model with 82 differential and 122 algebraic state variables
- Desired sampling time: 30 seconds.

Distillation Online Scenario

- System is in steady state, optimizer predicts constant trajectory:

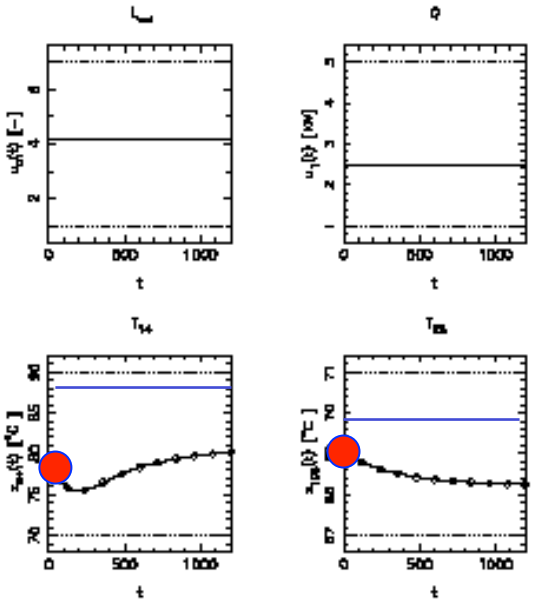


- **Suddenly**, system state x_0 is disturbed.
- What to do with optimizer?

Conventional Approach

- use offline method, e.g. MUSCOD-II with BFGS (Leineweber, 1999).
- initialize with **new** initial value x_0 and integrate system with **old** controls.
- iterate until convergence.

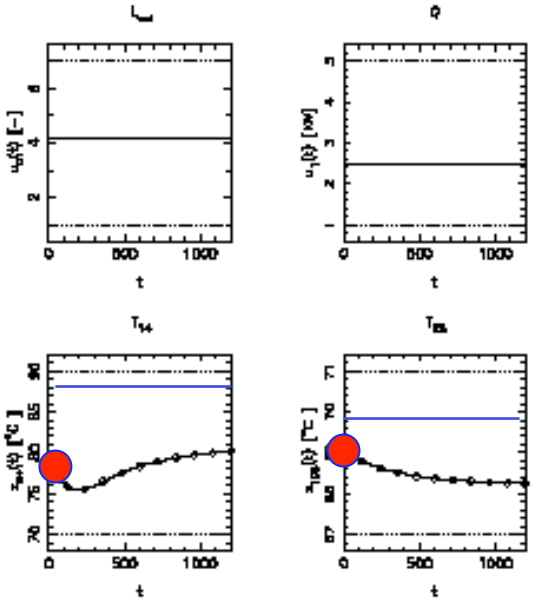
Initialization



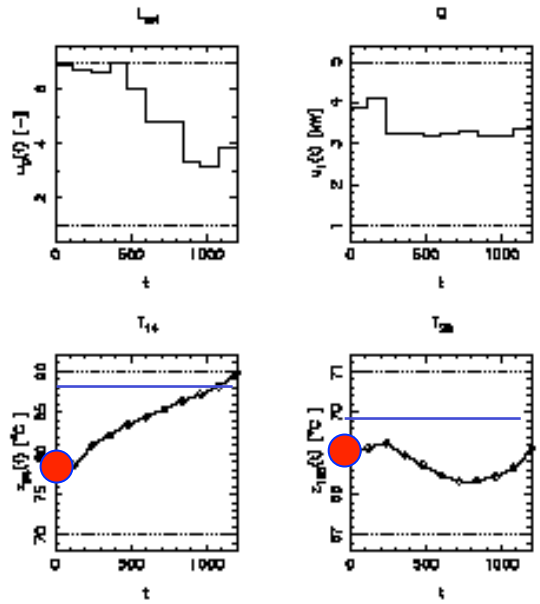
Conventional Approach

- use offline method, e.g. MUSCOD-II with BFGS (Leineweber, 1999).
- initialize with **new** initial value x_0 and integrate system with **old** controls.
- iterate until convergence.

Initialization



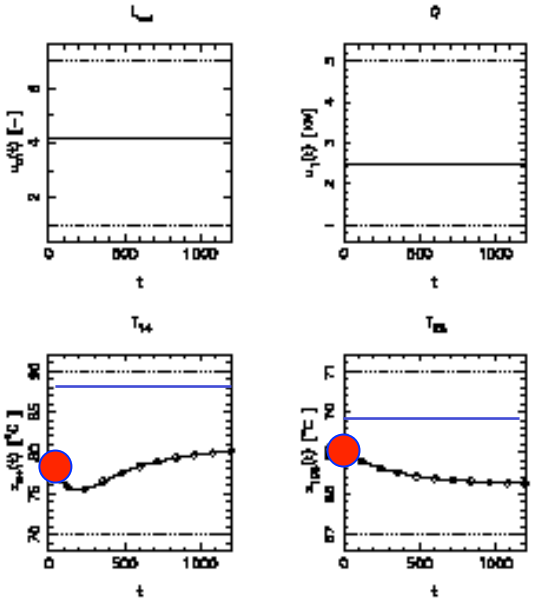
16th Iteration



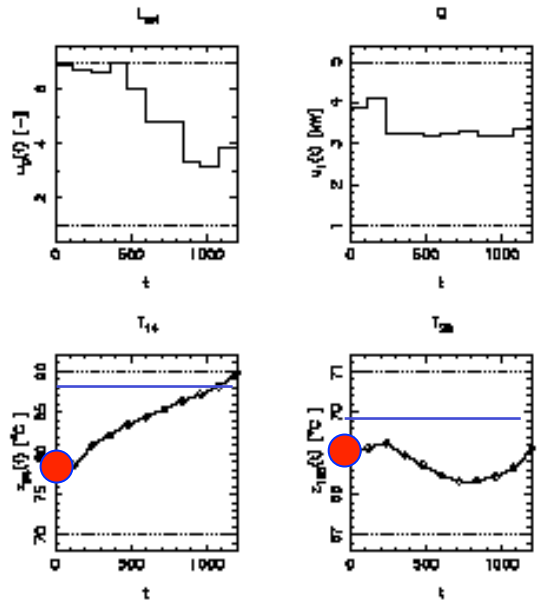
Conventional Approach

- use offline method, e.g. MUSCOD-II with BFGS (Leineweber, 1999).
- initialize with **new** initial value x_0 and integrate system with **old** controls.
- iterate until convergence.

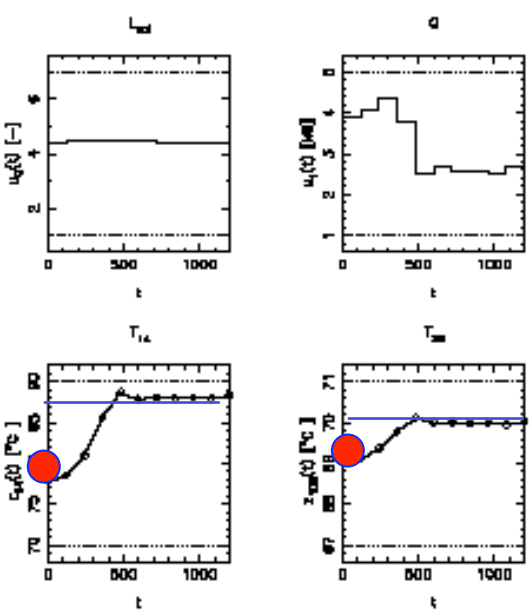
Initialization



16th Iteration



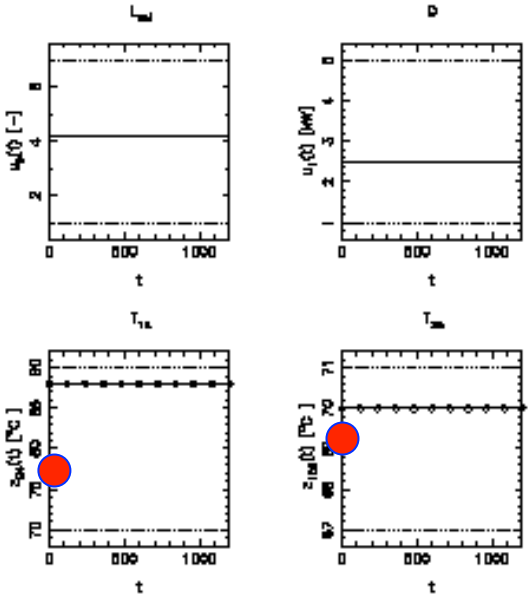
Solution (32nd Iteration



New Approach: Initial Value Embedding

- Initialize with **old** trajectory, accept violation of $s_0^x - x_0 = 0$

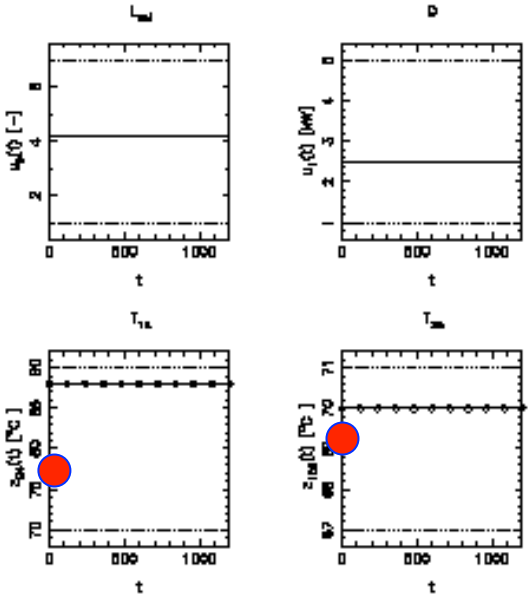
Initialization



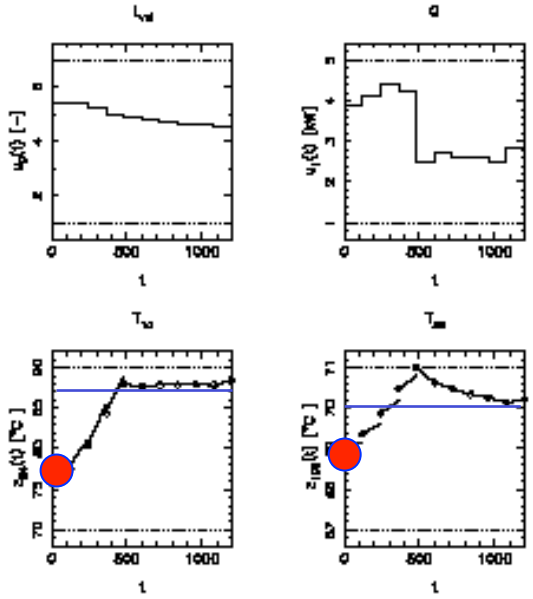
New Approach: Initial Value Embedding

- Initialize with **old** trajectory, accept violation of $s_0^x - x_0 = 0$

Initialization



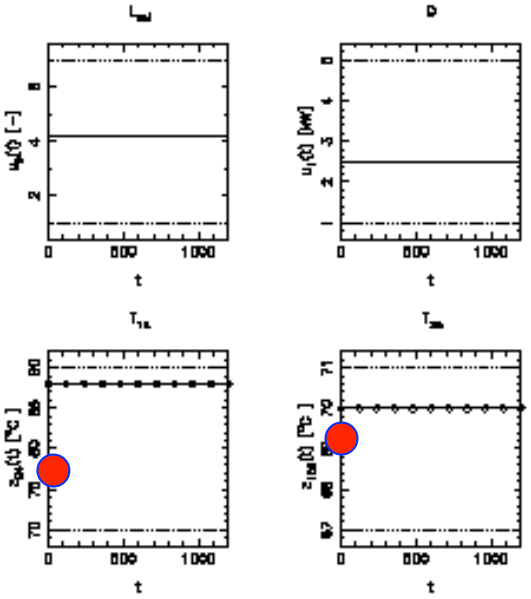
First Iteration



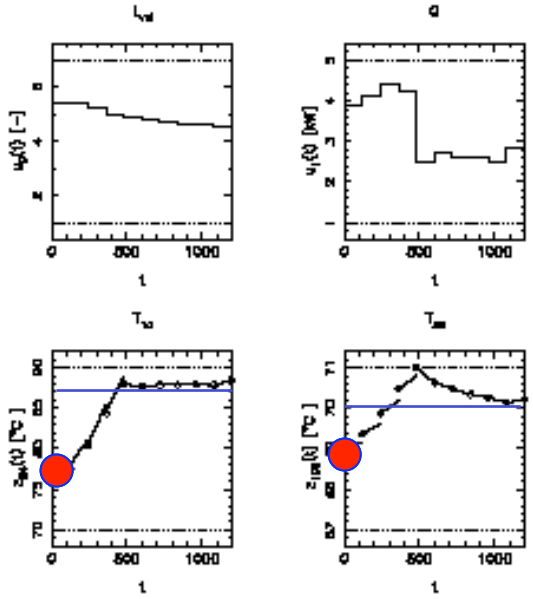
New Approach: Initial Value Embedding

- Initialize with **old** trajectory, accept violation of $s_0^x - x_0 = 0$

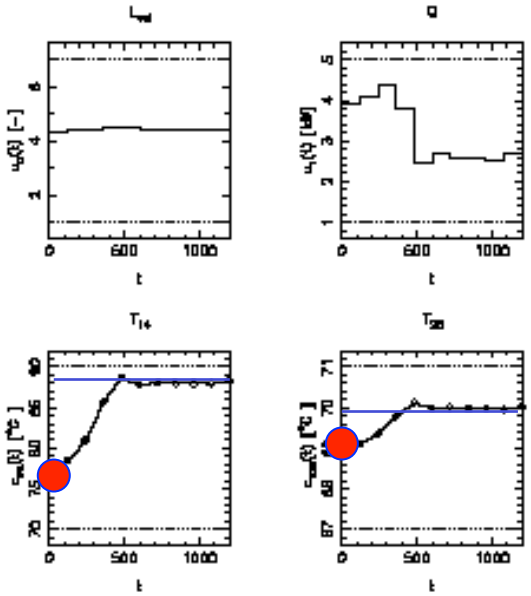
Initialization



First Iteration

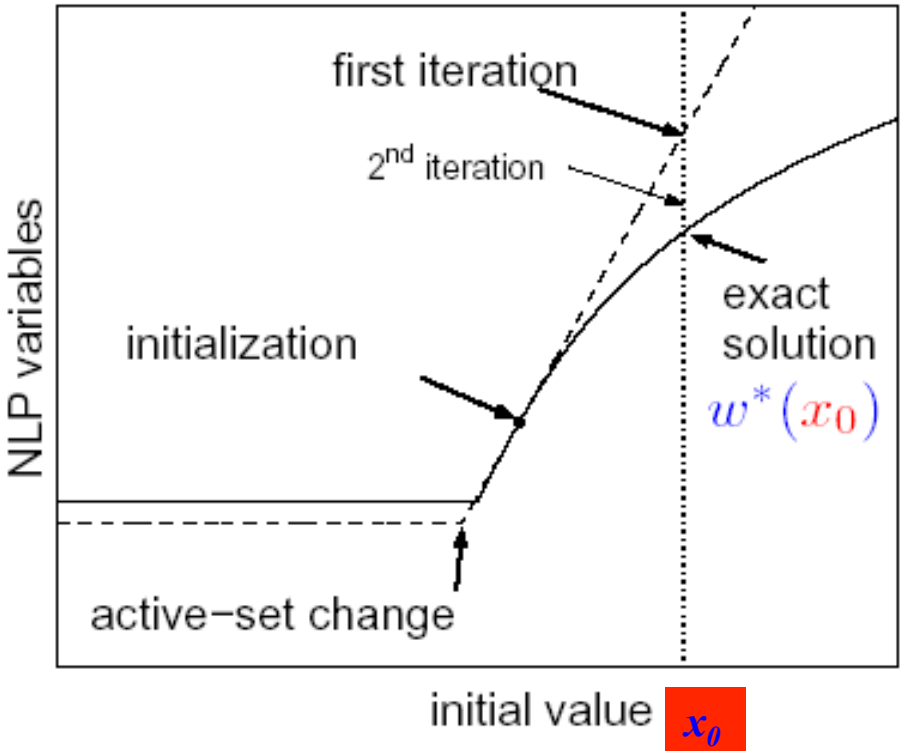


Solution (3rd Iteration)



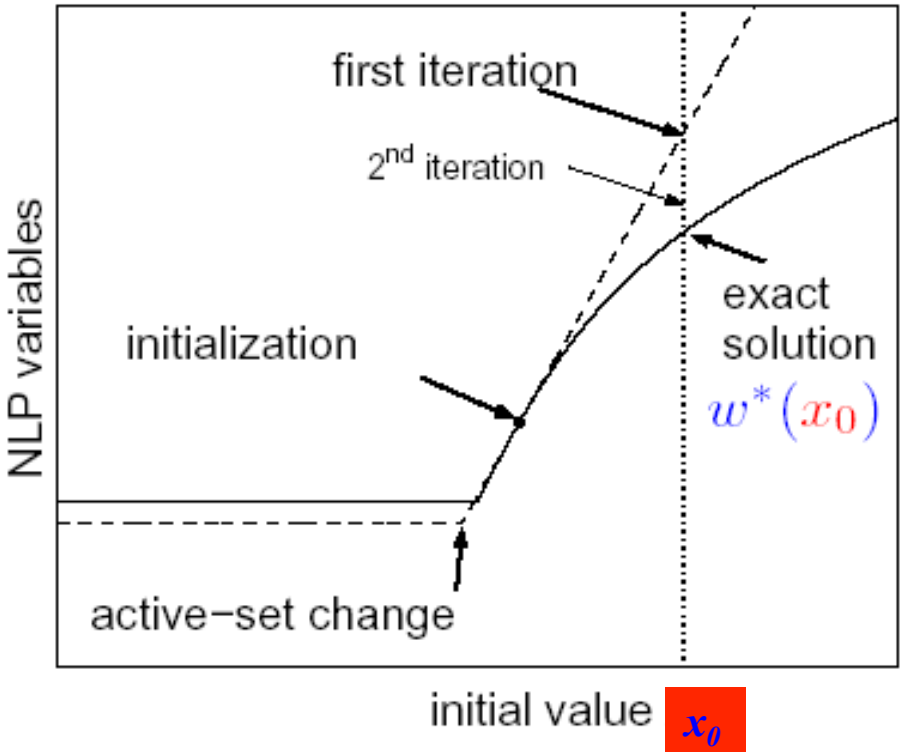
First iteration nearly solution!

Initial Value Embedding



- first iteration is tangential predictor for exact solution (for exact hessian SQP)
- also valid for active set changes
- derivative can be computed *before* x_0 is known: first iteration nearly without delay

Initial Value Embedding

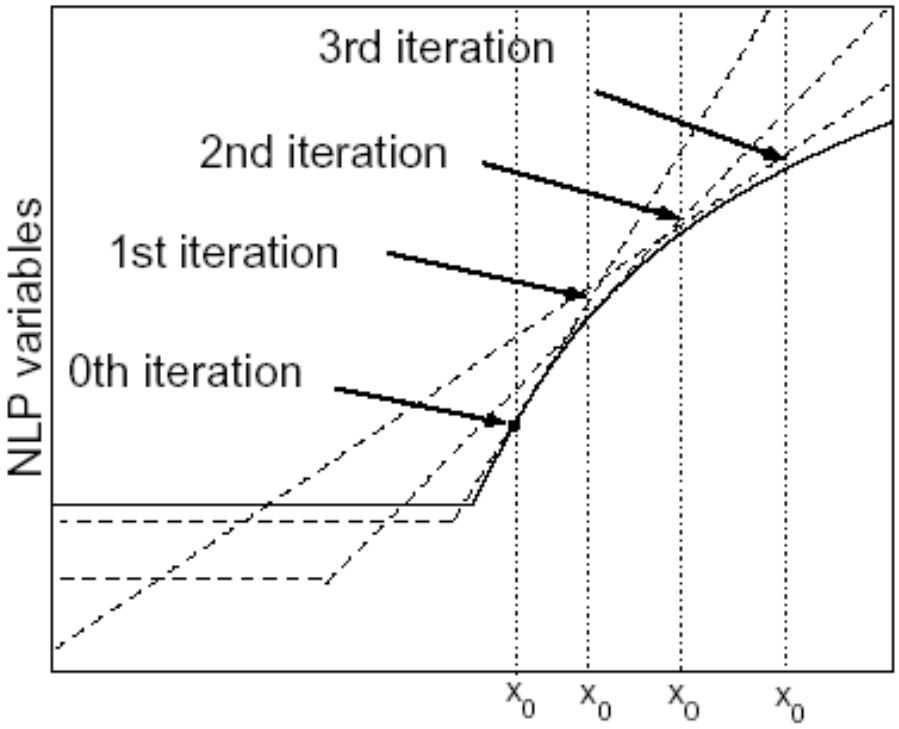


- first iteration is tangential predictor for exact solution (for exact hessian SQP)
- also valid for active set changes
- derivative can be computed *before* x_0 is known: first iteration nearly without delay

Why wait until convergence and do nothing in the meantime?

Real-Time Iterations [D. 2001]

Iterate, *while* problem is changing!



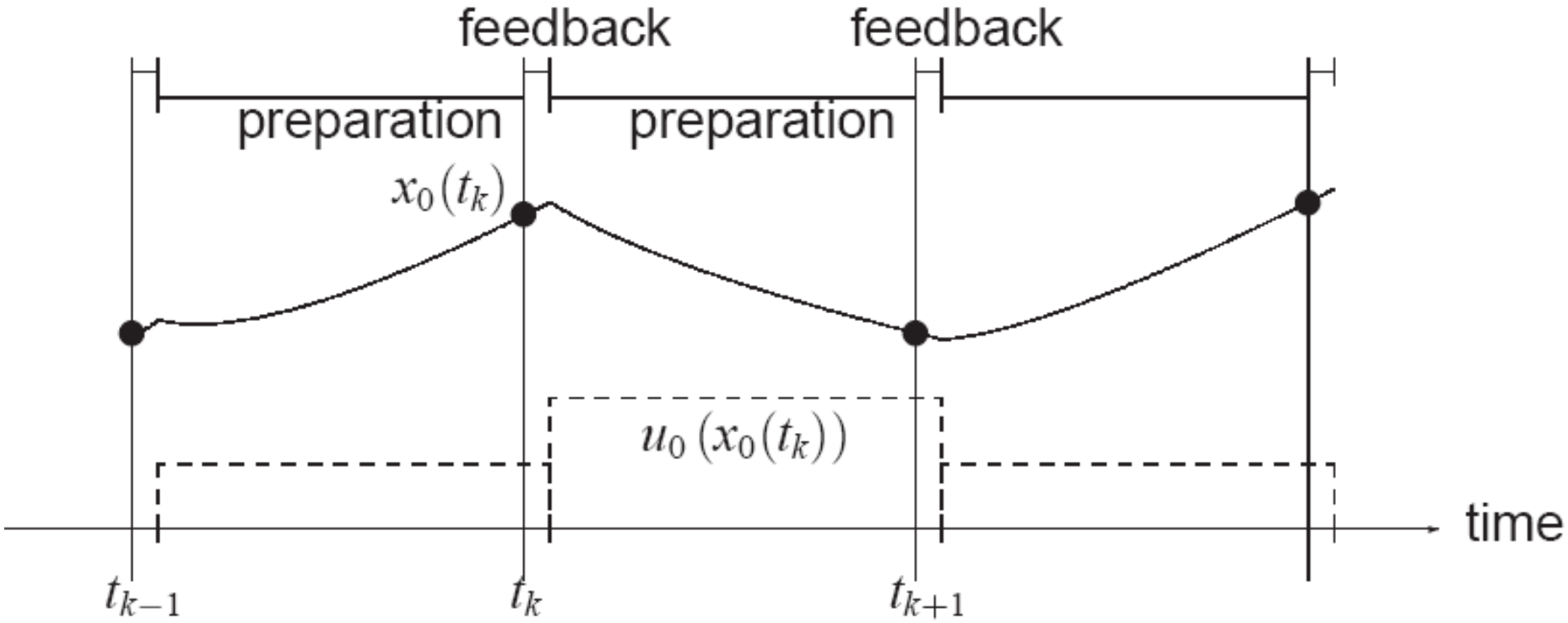
- tangential prediction after each change in x_0
- solution accuracy is increased with each iteration when x_0 changes little
- iterates stay close to solution manifold

Real-Time Iteration Algorithm:

1. **Preparation Step (costly):**
   Linearize system at current iterate, perform partial reduction and condensing of quadratic program.
2. **Feedback Step (short):**
   When new x_0 is known, solve condensed QP and implement control u_0 immediately.
   Complete SQP iteration. Go to 1.

- short cycle-duration (as **one** SQP iteration)
- negligible feedback delay ($\approx$ 1 % of cycle)
- nevertheless fully nonlinear optimization

Real-Time Iterations minimize feedback delay

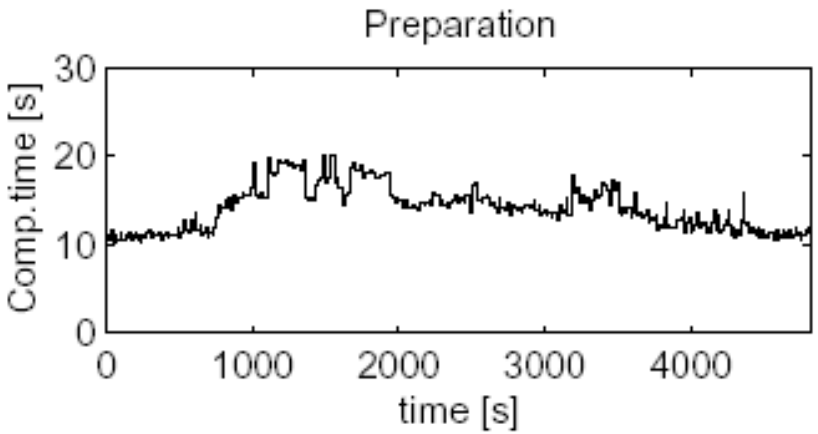


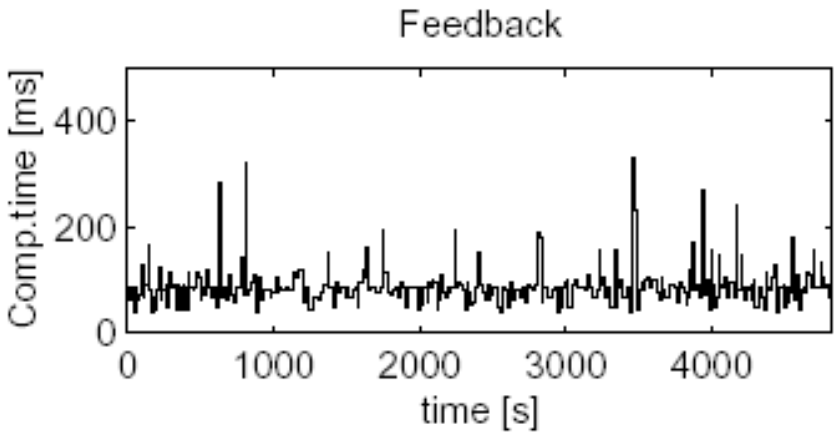
Realization at Distillation Column

[D., Findeisen, Schwarzkopf, Uslu, Allgöwer, Bock, Schlöder,2002]

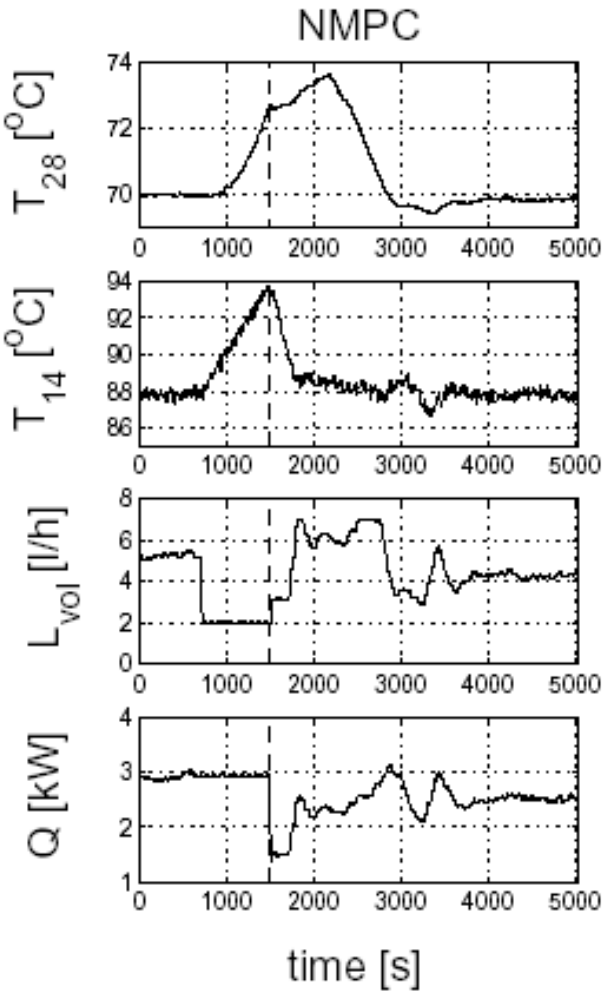
- Parameter estimation using dynamic experiments
- Online state estimation with Extended Kalman Filter variant, using only 3 temperature measurements to infer all 82 system states
- Implementation of estimator and optimizer on Linux Workstation.
- Communication with Process Control System via FTP all 10 seconds.
- Self-synchronizing processes.

Computation Times During Application



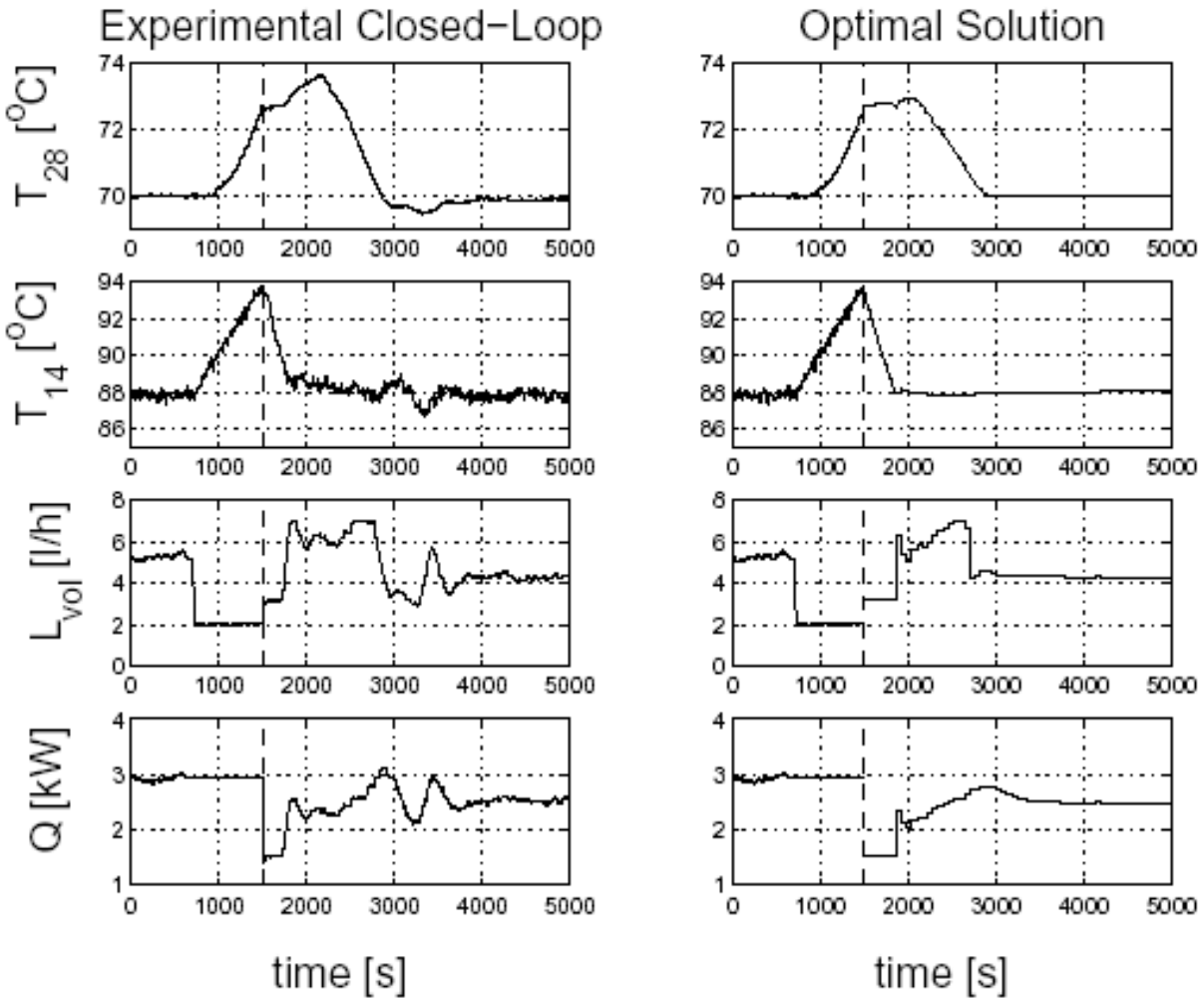


Large Disturbance (Heating), then NMPC



- Overheating by manual control
- NMPC only starts at $t = 1500$ s
- PI-controller not implementable, as disturbance too large (valve saturation)
- NMPC: at start control bound active $\Rightarrow T_{28}$ rises further
- Disturbance attenuated after half an hour

Real vs. Theoretical Optimal Solution



ACADO Toolkit for Nonlinear MPC

with Joachim Ferreau and Boris Houska

Software for Nonlinear MPC: ACADO Toolkit

- ACADO = Automatic Control and Dynamic Optimization

- Open source (LGPL): www.acadotoolkit.org
- User interface close to mathematical syntax
- Self containedness: only need C++ compiler
- Focus on small but fast applications

- Implemented Algorithms for Medium to Large Scale Problems:
  - **Inexact SQP methods for large DAE systems**
  - **Lifted Newton methods for large dynamic systems with fixed initial values.**

Problem Classes in ACADO

- Optimal Control of Dynamic Systems (ODE/DAE)

$$
\begin{aligned}
&\underset{y(\cdot),u(\cdot),p,T}{\text{minimize}} && \int_0^T L(\tau, y(\tau), u(\tau), p)\,\mathrm{d}\tau \;+\; M(y(T), p) \\
&\text{subject to:} && \\
\forall t \in [0,T]: \quad & 0 &=\; & f(t, \dot{y}(t), y(t), u(t), p) \\
& 0 &=\; & r(y(0), y(T), p) \\
\forall t \in [0,T]: \quad & 0 &\geq\; & s(t, y(t), u(t), p)
\end{aligned}
$$

- Nonlinear Model Predictive Control
- Parameter Estimation and Optimum Experimental Design
- Robust Optimization
- **Automatic Code Generation for fast MPC applications**

Example for Code Generation ("Tiny" Scale)

```
DifferentialState    p,v,phi,omega;
Control              a;

Matrix Q = eye( 4 );
Matrix R = eye( 1 );

DifferentialEquation f;
f << dot(p)     == v;
f << dot(v)     == a;
f << dot(phi)   == omega;
f << dot(omega) == -g*sin(phi)
                   -a*cos(phi)-b*omega;

OCP ocp( 0.0, 5.0 ,10 );
ocp.minimizeLSQ( Q, R );

ocp.subjectTo( f );
ocp.subjectTo( -0.2 <= a <=  0.2 );

OptimizationAlgorithm algorithm(ocp);
algorithm.solve();
```

Algorithm: Gauss Newton
Real-Time Iterations

1 control input
10 control intervals
4 states

	CPU time	Percentage
Integration & sensitivities	34 μs	63 %
Condensing	11 μs	20 %
QP solution (with qpOASES)	5 μs	9 %
Remaining operations	< 5 μs	< 8 %
One complete real-time iteration	54 μs	100 %

Inexact SQP methods for large scale DAE

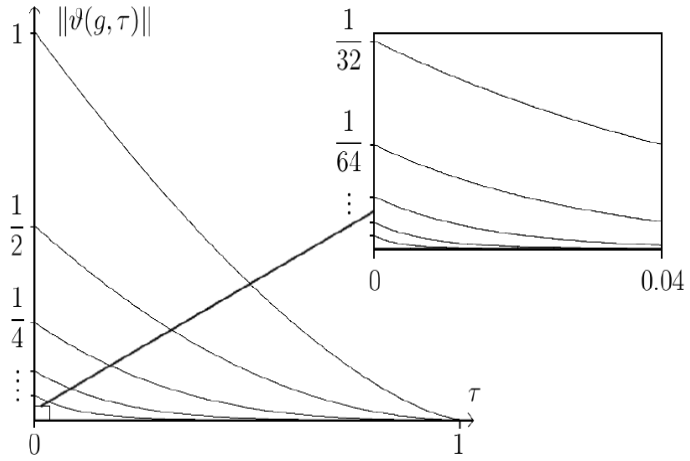
Problem: Computing sensitivities for large scale DAE's with many algebraic states is expensive

Idea: Relax the algebraic equation of the DAE [Bock 1987]

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}x_i(t) &= f(t, x_i(t), z_i(t), u_i, p) \\ 0 &= g(t, x_i(t), z_i(t), u_i, p) - \vartheta\left(\gamma_i, t-i\right) \\ \text{with} \quad x_i(0) &= s_i \ , \end{aligned}$$

Use a special relaxation function [Houska, Diehl 2010]

$$\vartheta(\gamma,\tau) := \begin{cases} \gamma\ (1-\tau)^{\frac{a+b\|\gamma\|}{\|\gamma\|}} & \text{if } \|\gamma\| \neq 0 \\ 0 & \text{otherwise} \end{cases}$$



Advantages of new DAE Relaxation

Can construct Multiple-Shooting inexact SQP method which converges q-quadratically.

No derivatives with respect to algebraic states needed.

No consistent initialization of the algebraic states within the DAE integrator needed.

Result for a Large Scale DAE model (distillation column with 82 differential and 122 algebraic states) in ACADO:

k	1	2	3	4	5
KKT-Tol:	$2.252 \cdot 10^{0}$	$4.230 \cdot 10^{-1}$	$7.34 \cdot 10^{-3}$	$2.32 \cdot 10^{-5}$	$1.52 \cdot 10^{-9}$

Computation Time: approx. 6 sec. per SQP step.

Lifted Newton Method

Question: Can we be even faster than 6 sec per SQP step?

Idea: Lifted Newton Method – reduce derivative computation to same load as single shooting

Option to turn on Lifted Newton in ACADO:

```
algorithm.set( DYNAMIC_SENSITIVITY,  FORWARD_SENSITIVITY_LIFTED );
```

Computation time for Stuttgart distillation column with Lifted Newton: approx 0.5 sec per SQP step

Modelica and Automatic Derivatives with CasADi

with **Joel Andersson**

CasADi

- CasADi
  - “Computer Algebra System for Automatic Differentiation”
  - Free (LGPL) open-source symbolic tool (www.casadi.org)
  - **Extends the NLP approach for OCP to shooting methods**
  - “Write a state-of-the-art multiple shooting code in 50 lines”

CasADi

The NLP Approach to Optimal Control

- Motivation: “The NLP approach”
  - Large scale OCP problems best solved with **direct methods**
    - E.g. single shooting, multiple shooting, collocation
  - Tools exist that take OCP problems in standard form…
    - ACADO Toolkit, MUSCOD-II, DIRCOL, DyOS
  - … but advanced users often prefer the “NLP approach”
    - “IPOPT-AMPL” approach
    - User responsible of reformulating OCP to NLP
    - Derivative information generated automatically
    - Formulate arbitrarily complex non-standard OCP
  - **Until now only for direct collocation!**

CasADi – NLP approach for Shooting Methods

Components of CasADi

- A computer algebra system for algebraic modeling
- Efficient, general implementation of AD
  - AD on **sparse, matrix-valued** computational graphs
  - Forward/adjoint mode
  - Generate new graphs for Jacobians/Hessians
- Efficient virtual machine for function/derivative evaluation
- Front-ends to C++, Python and Octave
- Smart interfaces to numerical codes, e.g.:
  - NLP solvers: IPOPT, KNITRO, (SNOPT, LiftOpt)
  - DAE integrators: Cvodes, Idas, GSL
    - Automatic generation of Jacobian information (for BDF)
    - Automatic formulation of sensitivity equations (fwd/adj)
- Symbolic model import from Modelica (via Jmodelica.org)

CasADi Code Example: Single Shooting in 30 lines

```
from casadi import *

# Declare variables (use simple, efficient DAG)
t = SX("t") # time
x=SX("x"); y=SX("y"); u=SX("u"); L=SX("cost")

# ODE right hand side function
f = [(1 - y*y)*x - y + u, x, x*x + y*y + u*u]
rhs = SXFunction([[t],[x,y,L],[u]],[f])

# Create an integrator (CVodes)
I = CVodesIntegrator(rhs)
I.setOption("abstol",1e-10) # abs. tolerance
I.setOption("reltol",1e-10) # rel. tolerance
I.setOption("steps_per_checkpoint",100)
I.init()

# All controls (use complex, general DAG)
NU = 20; U = MX("U",NU)

# The initial state (x=0, y=1, L=0)
X  = MX([0,1,0])

# Time horizon
T0 = MX(0);  TF = MX(20.0/NU)

# State derivative, algebraic state (not used)
XP = MX(); Z = MX()

# Build up a graph of integrator calls
for k in range(NU):
  [X,XP,Z] = I.call([T0,TF,X,U[k],XP,Z])

# Objective function: L(T)
F = MXFunction([U],[X[2]])

# Terminal constraints: 0<=[x(T);y(T)]<=0
G = MXFunction([U],[X[0:2]])

# Create NLP solver
solver = IpoptSolver(F,G)
solver.setOption("tol",1e-5)
solver.setOption("hessian_approximation", \
                 "limited-memory")
solver.setOption("max_iter",1000)
solver.init()

# Set bounds and initial guess
solver.setInput(NU*[-0.75], NLP_LBX)
solver.setInput(NU*[1.0],NLP_UBX)
solver.setInput(NU*[0.0],NLP_X_INIT)
solver.setInput([0,0],NLP_LBG)
solver.setInput([0,0],NLP_UBG)

# Solve the problem
solver.solve()
```

Outlook:
Multiple Shooting for Distributed Systems



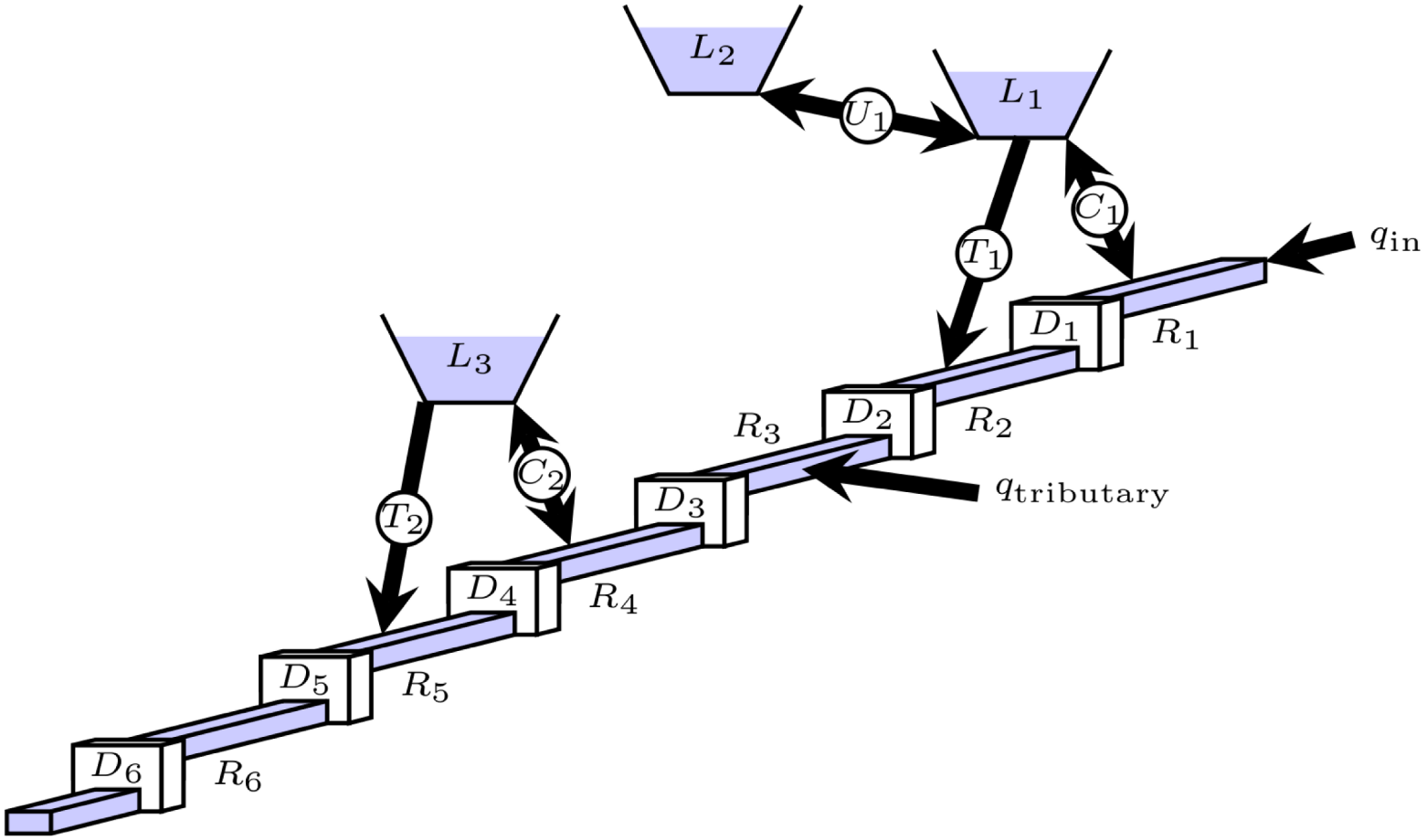
with

Attila Kozma, Carlo Savorgnan, Quoc Tran Dinh

HD–MPC

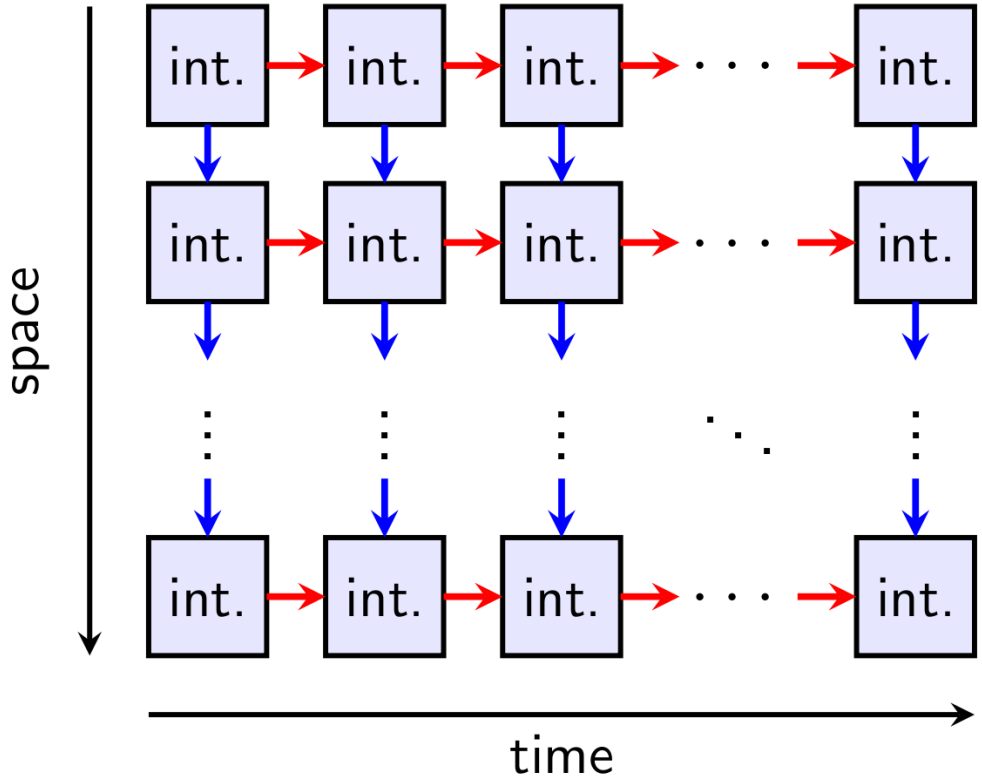
Multiple Shooting for Distributed Systems

MSD on Hydro Power Valley (HPV)



Multiple Shooting for Distributed Systems

Multiple Shooting in time AND SPACE



- discretized subsystem connections (polynomiasl)
- gaps between subsystems
- any complex topology

→ Talk Carlo Savorgnan

Large Scale QP algorithms

Decomposition by Lagrangian dual function

$$\min_{\underline{x}_1,\ldots,\underline{x}_N} \quad \sum_{i=1}^{N} \frac{1}{2}\underline{x}_i^T Q_i \underline{x}_i + \underline{c}_i^T \underline{x}_i$$

$$\text{s.t.} \quad H_i \underline{x}_i \leq \underline{d}_i \quad i = 1, \ldots, n$$

$$\sum_{i=1}^{N} A_i \underline{x}_i = \underline{b}$$

- Convex separable QP
- Coupling lin. equality

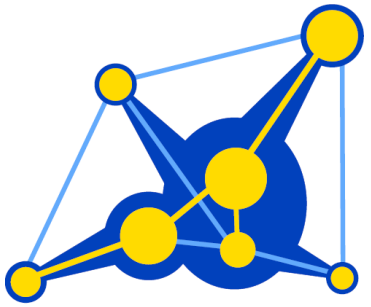
$$\max_{\underline{\lambda}} \quad \sum_{i=1}^{N} \underbrace{\left(\min_{\underline{x}_i} \quad \left(\frac{1}{2}\underline{x}_i^T Q_i \underline{x}_i + \left(\underline{c}_i^T + \underline{\lambda}^T A_i \right) \underline{x}_i - \underline{\lambda}^T \frac{\underline{b}}{N} \right) \right.}_{P_i(\underline{\lambda})}$$
$$\text{s.t.} \quad H_i \underline{x}_i \leq \underline{d}_i$$

- Two-level problem
- Low-level: parametric QPs (online act. set strat.)
- High-level: unconstr. problem with gradient avail. (fast gradient method)

Summary: Large Nonlinear MPC

OPTEC produces open-source (LGPL) software for academia and industry:

- qpOASES: linear MPC up to 0.2 MHz range (IPCOS, Hoerbiger, …)
- ACADO Toolkit: Nonlinear MPC up to 20 kHz (~4000 downloads)
- CasADi: “write your own optimal control solver” (used in JModelica)
- Distributed MPC → Talk C. Savorgnan



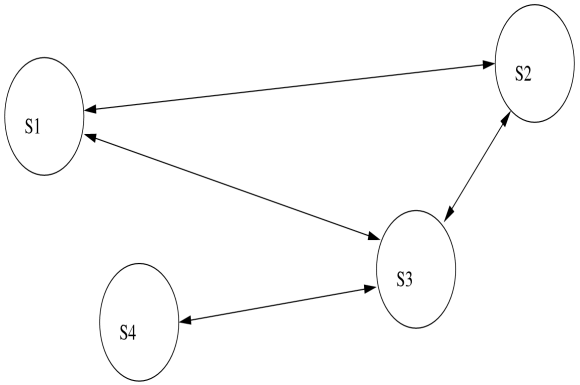
Appendix

Large-scale separable convex optimization (T.Quoc)

- Problem Statement

$$(\mathrm{CP}) \begin{cases} \min\limits_{x_1,\ldots,x_M} f(x) := \sum\limits_{i=1}^{M} f_i(x_i, \\ \text{s.t.} \sum\limits_{i=1}^{M} A_i x_i = b, \\ \qquad x_i \in X_i, \ i = 1, \ldots, M. \end{cases}$$

  - $f_i : \mathbb{R}^{n_i} \to \mathbb{R}$ - convex, possible nonsmooth
  - $X_i \subset \mathbb{R}^{n_i}$ – closed convex, bounded



- Examples
  - Large-scale LPs, QPs.
  - Optimization in networks, graph theory.
  - Multi-stage stochastic convex optimization.
  - Distributed MPC, etc.

- Aim:
  - Design distributed algorithms to solve (CP)

Main idea and algorithms

- Main idea: Combine three techniques
  - Lagrangian dual decomposition

$$d(y) = \sum_{i=1}^{M} d_i(y)$$

  - Smoothing technique via prox-functions

$$f_{\beta_2}(x) := \max_{y \in Y} \{\phi(x) + (Ax - b)^T y - \beta_2 p_Y(y)\}$$

$$d_{\beta_1}(y) := \min_{x \in X} \{f(x) + y^T (Ax - b) + \beta_1 p_X(x)\}$$

  - Excessive gap condition *[Nesterov2005]*

$$f_{\beta_2}(\bar{x}) \leq d_{\beta_1}(\bar{y})$$

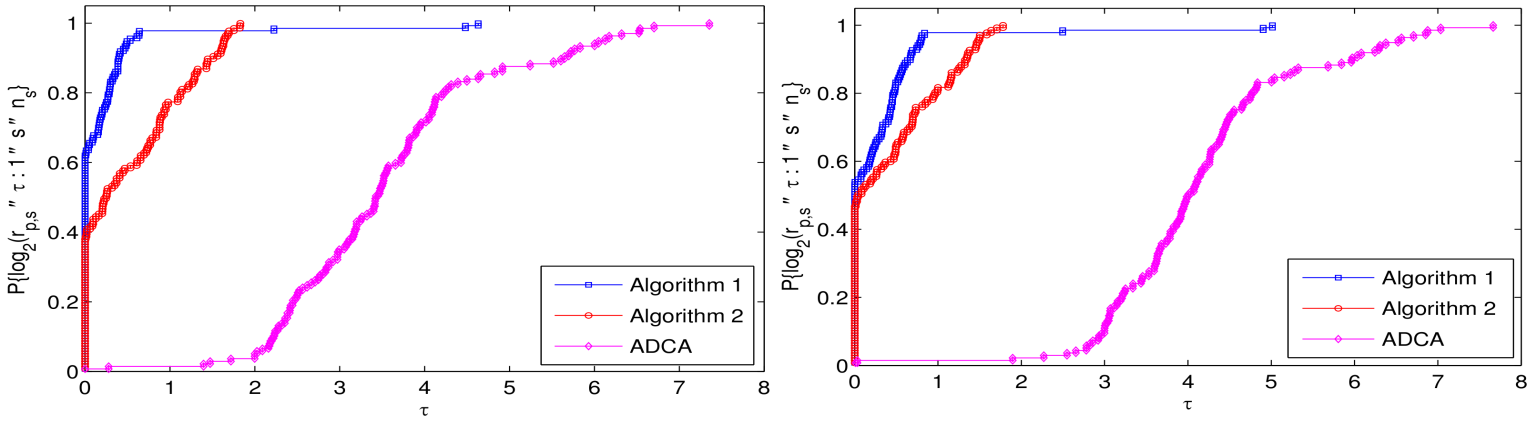
- Optimality and feasibility gaps

$$0 \leq \phi(\bar{x}) - d(\bar{y}) \leq \beta_1 D_o \quad \text{and} \quad ||A\bar{x} - b|| \leq \beta_2 D_f.$$

- Algorithm: two variants – primal update and switching update
  - Generate a sequence $\{(\bar{x}, \bar{y})\}$ such that it maintains the excessive gap condition, while controls β_1 and β_2 to zero.

Advantages and performance

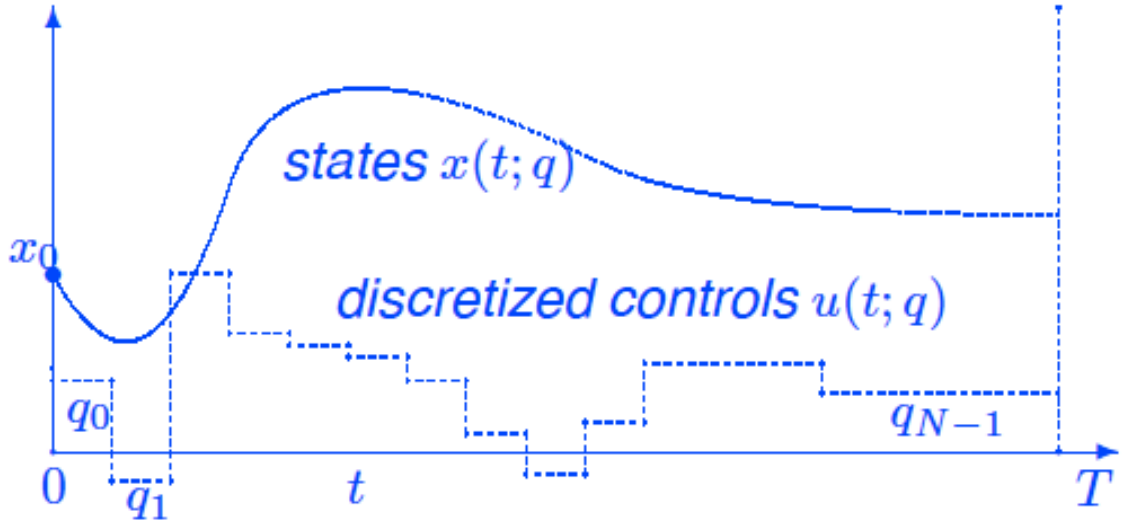
- Advantages
  - Convergence rate **O(1/k)**
  - Fast (compared to dual-fast gradient method *[Necoara2008]*, subgradient, augmented Lagrangian)
  - Numerical robustness
  - Highly distributed
- Numerical test: Large scale separable QP problems (dense)



Compare three difference algorithms: primal update, switching update, dual-fast gradient for solving random QPs (left – iterations, right – CPU time)

Direct Single Shooting [Hicks, Ray 1971; Sargent, Sullivan 1977]

Discretize controls $u(t)$ on fixed grid $0 = t_0 < t_1 < \ldots < t_N = T$, regard states $x(t)$ on $[0,T]$ as dependent variables.



Use numerical integration to obtain state as function $x(t;q)$ of finitely many control parameters $q = (q_0, q_1, \ldots, q_{N-1})$

NLP in Direct Single Shooting

After control discretization and numerical ODE solution, obtain NLP:

$$\underset{q}{\text{minimize}} \quad \int_0^T L(x(t;q),u(t;q))\,dt \;+\; E\left(x(T;q)\right)$$

subject to

$$\begin{aligned} h(x(t_i;q),u(t_i;q)) &\geq 0, \qquad i=0,\ldots,N, && \text{(discretized path constraints)} \\ r\left(x(T;q)\right) &\geq 0. && \text{(terminal constraints)} \end{aligned}$$

Solve with finite dimensional optimization solver, e.g. Sequential Quadratic Programming (SQP).

Solution by Standard SQP

Summarize problem as

$$\min_q F(q) \text{ s.t. } H(q) \geq 0.$$

Solve e.g. by Sequential Quadratic Programming (SQP), starting with guess q^0 for controls. $k := 0$

1. Evaluate $F(q^k), H(q^k)$ by ODE solution, and derivatives!

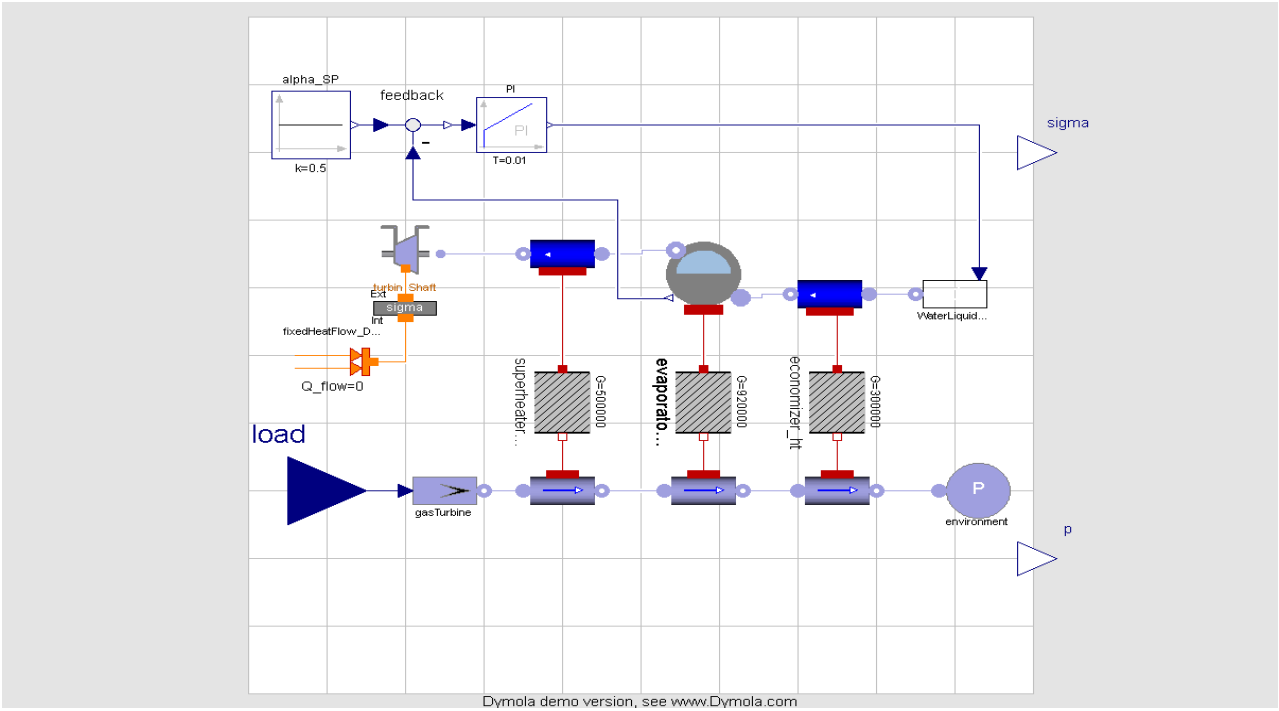
2. Compute correction Δq^k by solution of QP:

$$\min_{\Delta q} \nabla F(q_k)^T \Delta q + \frac{1}{2} \Delta q^T A^k \Delta q \text{ s.t. } H(q^k) + \nabla H(q^k)^T \Delta q \geq 0.$$

3. Perform step $q^{k+1} = q^k + \alpha_k \Delta q^k$ with step length α_k determined by line search.

CasADi – CC power plant

- **Application example: CC power plant**
  - Combined Cycle Power Plant Startup optimization
  - Model by F. Casella et.al.
  - Details: Modelica Conference 2011 (J. Andersson, J. Åkesson, F. Casella, M.Diehl)
- 6000+ lines of Modelica code: 10 differential 127 algebraic states



CasADi – CC power plant

Solution : CC power plant

- Compile to XML using JModelica.org compiler
- Use CasADi in Python to formulate collocation NLP
  - Direct collocation with Radau discretization
  - 50 finite elements
- Get NLP with 13849 variables and 13759 constraints
- Solve with IPOPT using MA57 linear solver
  - Use exact Hessian
  - Convergence after 51 IP iterations
  - **Total NLP solution time: 1.97 seconds**
    - **0.4 seconds in CasADi, rest in IPOPT internally**