Hierarchical and Distributed Optimization Methods

Moritz Diehl, Attila Kozma, Carlo Savorgnan

Optimization in Engineering Center OPTEC
and Electrical Engineering Department ESAT
K.U. Leuven, Belgium

IFAC WC Milano,
August 28, 2011

Overview

- Motivation for Centralized Computation
- Distributed Multiple Shooting Framework
- Adjoint Based SCP Methods, from Hierarchical to Distributed
- Software

Motivation for Hierarchical and Distributed MPC

Large-scale systems in engineering

- composed of multiple subsystems
- complex nonlinear dynamics and
- mutual influences

E.g. river networks, chemical production sites, airflow in buildings.

How to compute optimal controls e.g. for transients?

Two Central Observations on distributed MPC

(1) For cooperative model predictive control, we ideally want to solve **one large centralized MPC problem**.

Reasons for distributed setup:

- Robustness and easier reconfigurability
- Distribution of data and model maintenance
- Parallel computations (ideally, solution time independendent of size)
- Hope that less communication is needed than in centralized setting

(2) Most distributed MPC methods work iteratively and focus on parallelizing each iteration. But even if solution time for each iteration is independent of size, the convergence speed mostly deteriorates with size of the problem (usually linear or sublinear rates).

Distributed computation and communication time might be much higher than for one centralized solution, i.e. many processors together working very hard can be slower than one single one!

(Interlude: 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)

(Runtime Comparison in Our Initial Work)

Solve large distributed quadratic program with 100 subsystems on 100 CPUs, using different dual decomposition methods:

Wall clock:

δ	Nesterov	Gradient
10^{-3}	0:55	02:58
10^{-4}	1:55	03:59
10^{-5}	2:52	04:56
10^{-6}	3:29	05:52

Same problem takes **0:03 seconds on a single CPU** when solved with a sparse IP method (OOQP from S. Wright).

Problem of all gradient methods: no second order information, slow linear convergence. Better parallelize IP solver!

Can simulation efficiently be parallelized ?

Assumption: simulators for individual subsystems exist

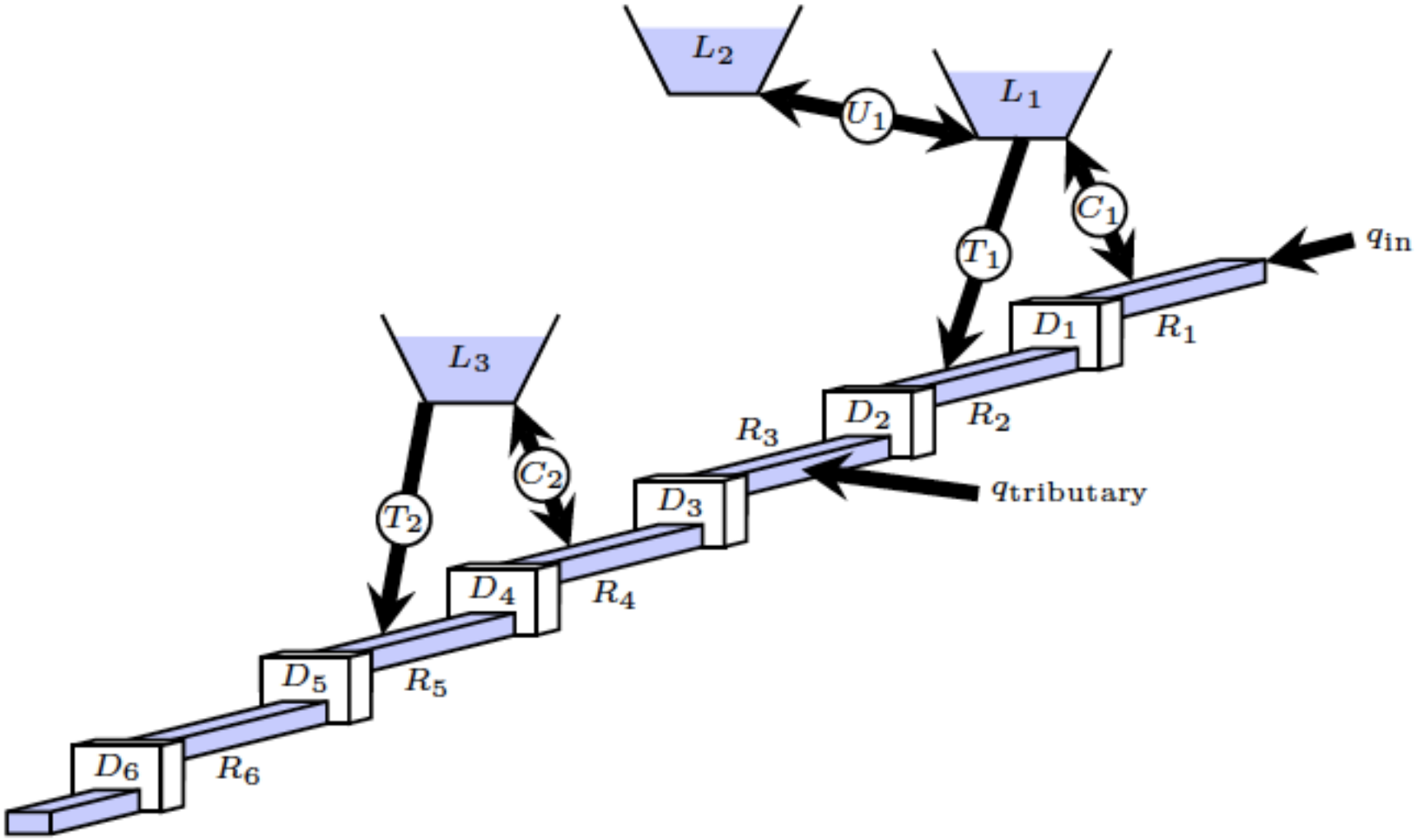
- use their own adaptive numerical integration schemes
- based on possibly different modelling languages
- can provide derivatives in forward and reverse mode (not yet standard, but provided e.g. by SUNDIALS, DASPK, DAESOL-II, ACADO Integrators, ...)

Example: Hydro Power Valley (HPV) Benchmark

River reaches connected by dams and hydro power units.
NMPC control aims:

- strictly respect level constraints
- match total power demand
- keep levels as constant as possible

HPV consists of 8 coupled subsystems



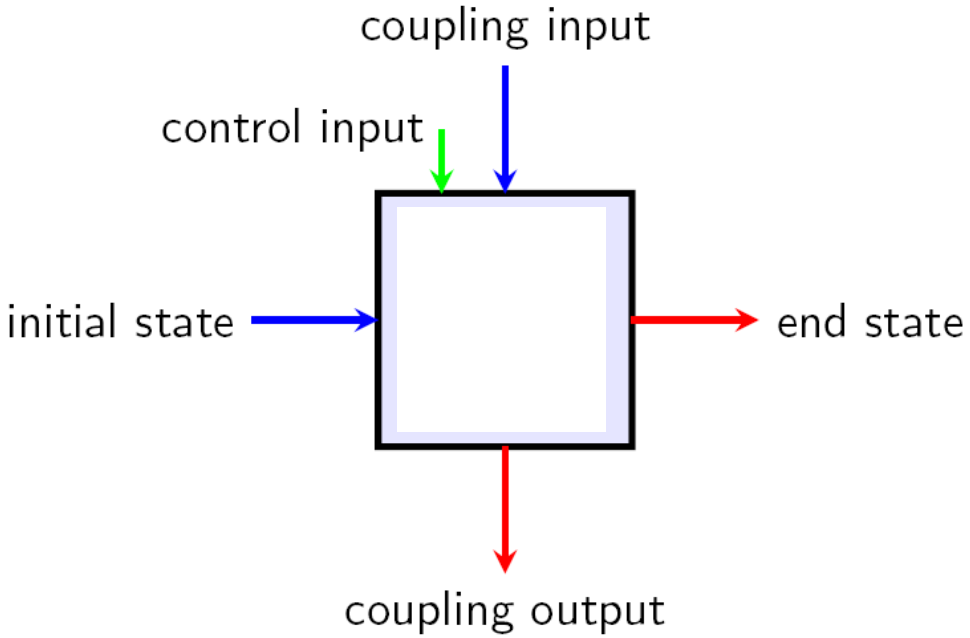
Hydro Power Valley (HPV)

Water flow in reaches modeled by Saint Venant PDE:

$$\begin{cases} \dfrac{\partial Q(z,t)}{\partial z} + w\dfrac{\partial H(z,t)}{\partial t} = 0 \\ \dfrac{1}{gw}\dfrac{\partial}{\partial t}\left(\dfrac{Q(z,t)}{H(z,t)}\right) + \dfrac{1}{2gw^2}\dfrac{\partial}{\partial z}\left(\dfrac{Q^2(t,z)}{H^2(t,z)}\right) + \dfrac{\partial H(t,z)}{\partial z} + I_f(z) - I_0 = 0 \end{cases}$$

Transform PDE into ODE by spatial discretization.

The "Simulation Box" (e.g. one reach of HPV)



Centralized Optimal Control

$$
\begin{aligned}
\min_{\substack{x,u,z,\\ y,e}} \quad & \int_0^T \ell(e(t))dt + \sum_{i=1}^{M} \int_0^T \ell^i(x^i(t), u^i(t), z^i(t))dt \\
\text{s.t.} \quad & \dot{x}^i(t) = f^i(x^i(t), u^i(t), z^i(t)) \\
& y^i(t) = g^i(x^i(t), u^i(t), z^i(t)) \\
& x^i(0) = \bar{x}_0^i \\
& z^i(t) = \textstyle\sum_{j=1}^{M} A_{ij} y^j(t) \\
& e(t) = r(t) + \textstyle\sum_{i=1}^{M} B^i y^i(t) \\
& p^i(x^i(t), u^i(t)) \geq 0, \quad q(e(t)) \geq 0 \quad t \in [0, T]
\end{aligned}
$$

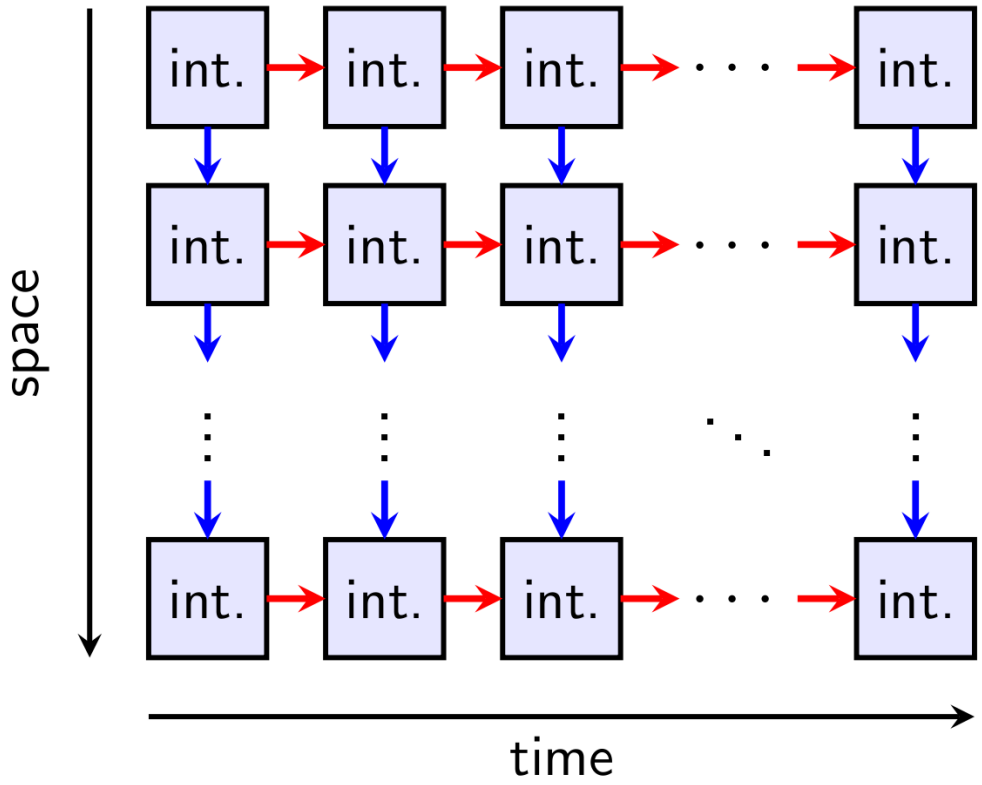
Key idea: The signals $z^i(t)$, $y^i(t)$ and $e(t)$ can be represented as a linear combination of orthogonal polynomials.

Distributed Multiple Shooting yields sparse NLP

$$
\begin{aligned}
\min_{\substack{u_n^i, x_n^i, \mathbf{z}_n^i, \\ \mathbf{y}_n^i, \mathbf{e}_n}} \quad & \sum_{n=0}^{N-1} \left(L_n(\mathbf{e}_n) + \sum_{i=1}^{M} L_n^i(x_n^i, u_n^i, \mathbf{z}_n^i) \right) \\
\text{s.t} \quad & x_{n+1}^i = F_n^i(x_n^i, u_n^i, \mathbf{z}_n^i) \qquad n = 0, \ldots, N-1 \\
& \mathbf{y}_n^i = G_n^i(x_n^i, u_n^i, \mathbf{z}_n^i) \qquad n = 0, \ldots, N-1 \\
& x_0^i = \bar{x}_0^i \\
& \mathbf{z}_n^i = \textstyle\sum_{i=1}^{M} A_{ij} \mathbf{y}_n^j \\
& \mathbf{e}_n = \mathbf{r}_n + \textstyle\sum_{i=1}^{M} B_{ij} \mathbf{y}_n^j \\
& p^i(x_n^i, u_n^i) \geq 0, \qquad Q_n(\mathbf{e}_n) \geq 0
\end{aligned}
$$

Distributed Multiple Shooting

Multiple Shooting [Bock and Plitt 1984], but in time AND SPACE



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

Large Scale Nonlinear Program (NLP)

Each simulation box $x_i = \phi_i(X_i, u_i)$ also evaluates an objective $f_i(X_i, u_i)$ and inequality constraints $g_i(X_i, u_i)$.

x_i = output of each simulation box.

X_i = Input, lin. combination of other outputs

$$\begin{aligned} \text{minimize}_{x,u} \quad & \sum_{i=1}^{N} f_i(X_i, u_i) \\ \text{subject to} \quad & \phi_i(X_i, u_i) - x_i = 0, \\ & g_i(X_i, u_i) \leq 0, \quad i = 1, \ldots, N. \end{aligned}$$

Note: coupling constraints only feasible in solution!

***Simultaneous* method for simulation and optimization.**

Sequential Convex Programming (SCP)

Assuming f_i, g_i convex and known to central optimizer, can linearize simulation boxes at linearization points $\bar{X}_i, \bar{u}_i$.

$$\text{minimize}_{x,u} \quad \sum_{i=1}^{N} f_i(X_i, u_i)$$

$$\text{subject to} \quad \phi_i(\bar{X}_i, \bar{u}_i) + \frac{\partial \phi_i(\bar{X}_i, \bar{u}_i)}{\partial (X, u)} \begin{bmatrix} X_i - \bar{X}_i \\ u_i - \bar{u}_i \end{bmatrix} - x_i = 0$$

$$g_i(X_i, u_i) \leq 0, \; i \in [1, N].$$

Iteratively solving linearized convex problems for obtaining the next linearization point yields a generalization of SQP, **Sequential Convex Programming (SCP)**. Can prove linear convergence towards local minima [Necoara et al, CDC, 2009], [T. D. Quoc and MD, BFG, 2010].

Adjoint based SCP Method

Approximate $\frac{\partial \phi_i(\bar{X}_i, \bar{u}_i)}{\partial (X,u)}$ by cheaper A_i. Add gradient correction to objective.

$$\text{minimize}_{x,u} \quad \sum_{i=1}^{N} f_i(X_i, u_i) + [X_i^T | u_i^T] \frac{\partial \phi_i(\bar{X}_i, \bar{u}_i)}{\partial (X,u)}^T \bar{\lambda}_i$$

$$\text{subject to} \quad \phi_i(\bar{X}_i, \bar{u}_i) + A_i \begin{bmatrix} X_i - \bar{X}_i \\ u_i - \bar{u}_i \end{bmatrix} - x_i = 0,$$

$$g_i(X_i, u_i) \leq 0, \; i \in [1, N].$$

Solution x^*, u^* and equality multipliers δ^* yield next linearization point $\bar{x}^+, \bar{u}^+$ and multiplier guess, $\bar{\lambda}^+ = \bar{\lambda} + \delta^*$.
Linear convergence proven [D., Walther, Bock, Kostina, OMS, 2009], [Quoc et al. 2010].

Why are inexact derivatives interesting ?

- derivative $\frac{\partial \phi_i(\bar{X}_i, \bar{u}_i)}{\partial (X,u)}$ is a large dense matrix, expensive to compute
- often, only few strongly coupling variables X_i^A in $X_i = (X_i^A, X_i^B)$, so can cheaply approximate derivative:

$$\left[\frac{\partial \phi_i}{\partial X_i^A} \;\middle|\; \frac{\partial \phi_i}{\partial X_i^B} \;\middle|\; \frac{\partial \phi_i}{\partial u_i} \right] \approx \left[\frac{\partial \phi_i}{\partial X_i^A} \;\middle|\; 0 \;\middle|\; \frac{\partial \phi_i}{\partial u_i} \right] =: A_i$$

- evaluate gradient correction $\frac{\partial \phi_i(\bar{X}_i, \bar{u}_i)}{\partial (X,u)}^T \bar{\lambda}_i$ by reverse differentiation, only 4 times more expensive than simulation $\phi_i(X_i, u_i)$. One single **extended simulation box** call.
- Less communication: variables x^B and multipliers λ^B only passed between child and parent nodes. Central optimizer works with **aggregate model** in x^A and u only.

Variant: left part of A_i = 0, get completely distributed convex subproblems!

Adjoint SCP: both Hierarchical and Distributed

- Exact SCP: all coordination work done by central agent who solves convex subproblems. 100% hierarchical.
- Adjoint based SCP with partially zero derivative matrices A_i : only most influential variables coordinated by central agent, fine interactions are exchanged locally.
- Adjoint based SCP with completely zero derivative matrices A_i : all information is exchanged locally, convex problem decomposes, no central agent necessary: 100% distributed.

Trade-off: convergence speed vs distributed solution

Overview

- Motivation for Centralized Computation
- Distributed Multiple Shooting Framework
- Adjoint Based SCP Methods, from Hierarchical to Distributed
- **Software**

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

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	$34\ \mu$s	63 %
Condensing	$11\ \mu$s	20 %
QP solution (with qpOASES)	$5\ \mu$s	9 %
Remaining operations	$< 5\ \mu$s	< 8 %
One complete real-time iteration	$54\ \mu$s	100 %

NMPC with 200 kHz possible !

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

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/CVODES for Sensitivies of HPV subsystem

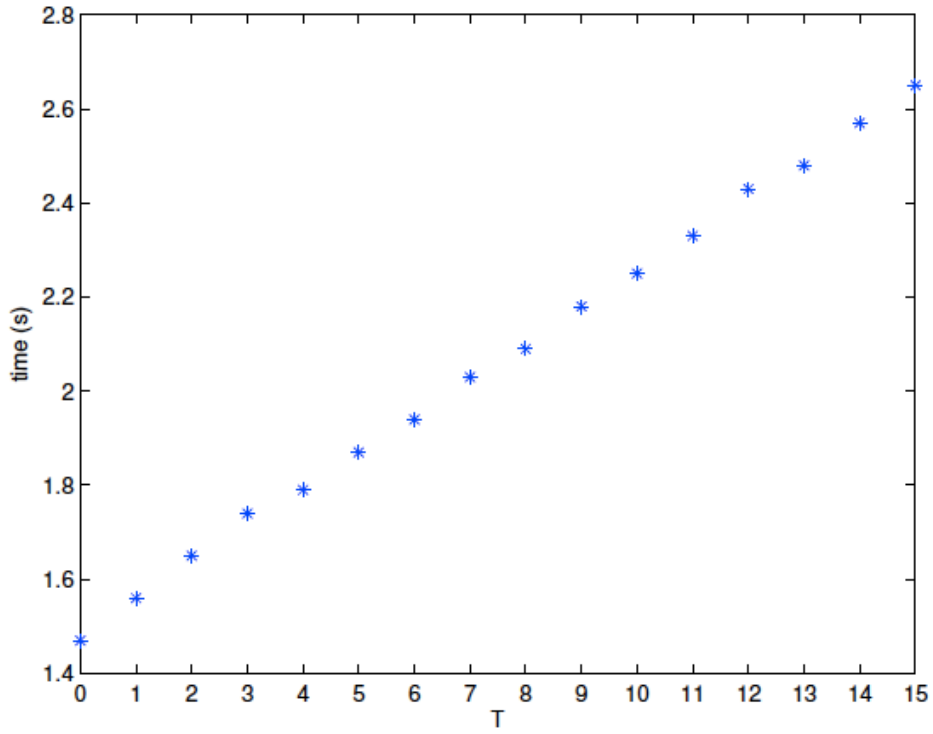


Fig. 4. Time required to integrate and linearize a subsystem dynamics for an time interval of 30 minutes using $S = 15$.

For full problem:

- Total: 48 time intervals, 8 subsystems = 384 simulation boxes
- Sensitivity Integration of full system on full horizon would take 1630 sec
- Compare this to 1.5 up to 2.7 sec per simulation box.

0 = only forward w.r.t. controls and adjoint, fully distributed

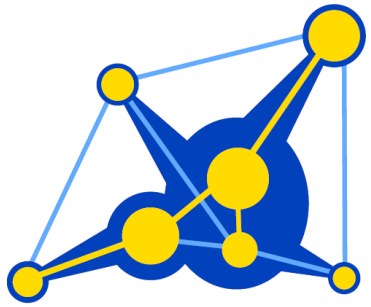
15 = all forward derivatives, full space exact SCP

Summary: Large Nonlinear MPC

- In cooperative MPC we want to solve centralized optimization problems, and centralized algorithms might be more efficient in both time and communication than distributed ones
- Distributed Multiple Shooting (DMS) is a way to parallelize simulation and sensitivity generation
- Adjoint based SCP Algorithms for DMS allow many variants between fully hierarchical and fully distributed algorithms

Software (LGPL):

- ACADO Toolkit and code generation allow fast nonlinear MPC for small problems (e.g. 200 kHz for 4 states)
- CasADi allows one to easily couple integrators and optimizers and setup e.g. distributed multiple shooting
  → Talk Attila Kozma, Monday, 11:20, room Vito



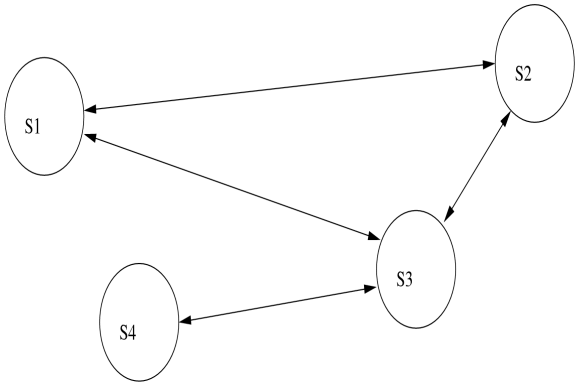
Appendix

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]*

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

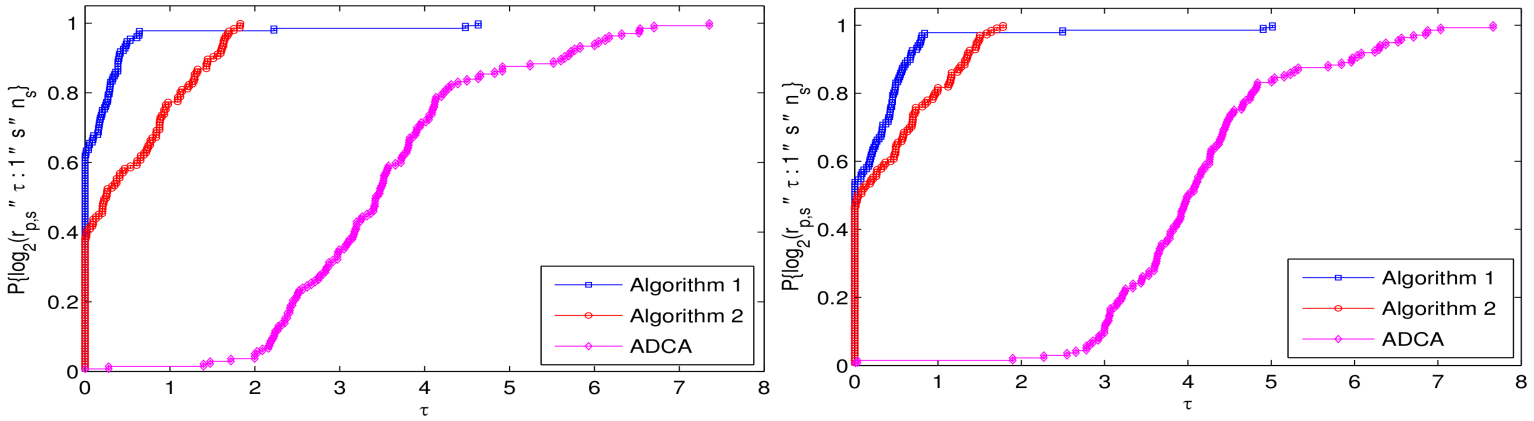
- Optimality and feasibility gaps

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

- 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)

Coupling between subsystems

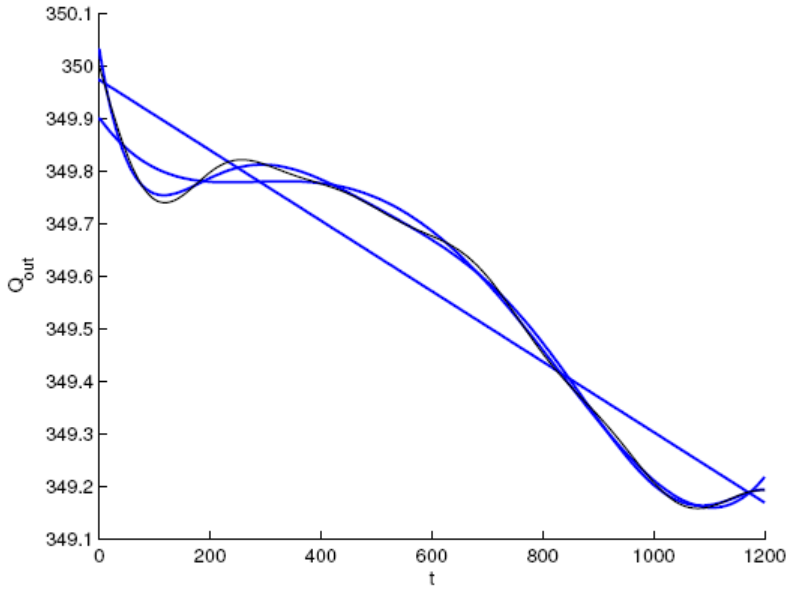
Dynamics of subsystems are coupled via in-/output profiles of "coupling variables". Infinite dimensional coupling.

Coupling between subsystems

Dynamics of subsystems are coupled via in-/output profiles of "coupling variables". Infinite dimensional coupling.

Can approximate coupling profile by orthogonal polynomials:

$$\int_a^b P_i(t)P_j(t)dt = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{otherwise} \end{cases}$$



Approximation of typical output water discharge profile (black) by polynomials of degree 1, 4 and 7.

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()
```