The acados software package
Fast, embeddable solvers for nonlinear optimal control

Jonathan Frey and Moritz Diehl

Systems Control and Optimization Laboratory (syscop)

acados workshop @ ELO-X fall school

October 5, 2022
Workshop Outline

- Presentation
  - Introduction
  - acados optimal control problem formulation
  - Overview on acados
  - QP solvers
  - integration methods
  - Python interface

- Interactive Exercise / Demo Session
Real world optimal control applications with fast dynamics, nonlinear optimal control problem formulations, strict hardware limitations require tailored high-performance algorithms.

acados implements such algorithms based on Sequential Quadratic Programming (SQP) Real-Time Iteration (RTI)

Application projects include
- Wind turbines
- Drones
- Race cars
- Driving assistance systems
- Electric drives (PMSM)
Continuous-time optimal control problem (OCP):

\[
\begin{align*}
\text{minimize} & \quad \int_{t=0}^{T} \ell(x(t), z(t), u(t)) \, dt + M(x(T)) \\
\text{subject to} & \quad x(0) = \bar{x}_0, \\
& \quad 0 = f(\dot{x}(t), x(t), z(t), u(t)), \quad t \in [0, T], \\
& \quad 0 \geq g(x(t), z(t), u(t)), \quad t \in [0, T].
\end{align*}
\]

(1)

In MPC, instances of these problems are solved repeatedly, with current state \( \bar{x}_0 \).
OCP structured NLP handled in acados

\[
\text{minimize} \quad \sum_{k=0}^{N-1} l_k(x_k, u_k, z_k) + M(x_N) + \sum_{k=0}^{N} \rho_k(s_k) \\
\text{subject to} \quad \begin{bmatrix} x_{k+1} \\ z_k \end{bmatrix} = \phi_k(x_k, u_k), \quad k = 0, \ldots, N - 1, \\
0 \geq g_k(x_k, z_k, u_k) - J_{s,k}s_k \quad k = 0, \ldots, N - 1, \\
0 \geq g_N(x_N) - J_{s,N}s_N, \\
0 \leq s_k \quad k = 0, \ldots, N.
\]

- \(\phi_k\) – discrete time dynamics on \([t_k, t_{k+1}]\) – typically acados integrator from ODE or DAE
- \(l_k\) discrete version of Lagrange cost term \(\ell\) on \([t_k, t_{k+1}]\)
- slack variables \(s_k\) separate from controls – handled efficiently
- inequality constraints \(g_k\)
- problem functions can vary stage wise in \(\mathbb{C}\)
- from high-level interfaces
  - initial and terminal shooting node handled separately – MHE support
  - parameters can be varied conveniently
- more detailed problem formulation can be found here.
Philosophy & History

- Solvers and interfaces for
  - OCP structured NLP (2)
  - Initial value problems for ODEs and DAEs – integrators
- Exploit block-sparse structure inherent in OCP / MHE formulation → specialized solver
- Successor of the ACADO Toolkit
  - Code generation for all parts of the SQP method
- Principles of acados
  - efficiency – BLASFE0, HPIPM, C
  - flexibility – general formulation
  - modularity – encapsulation
  - portability – self-contained C library with little dependencies
- Model functions code generation using CasADi
- Problem formulation in high-level interface (Python, MATLAB, Octave)
- Generate corresponding C code for problem specific solver
  - uses only acados C interface
  - first developed in Python interface
  - used for S-function generation – Simulink interface
The interplay between the acados dependencies, the ‘core’ C library and its interfaces.

- **BLASFE0**: Basic Linear Algebra for Embedded Optimization
- **HPIPM**: High-Performance Interior Point Method
# QP solver types and sparsity – an overview

<table>
<thead>
<tr>
<th></th>
<th>Active-Set</th>
<th>Interior-Point</th>
<th>First-Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense</td>
<td>qpOASES, DAQP</td>
<td>HPIPM</td>
<td></td>
</tr>
<tr>
<td>sparse</td>
<td>[PRESAS]</td>
<td>CVXGEN, OOQP</td>
<td>FiOrdOs, OSQP</td>
</tr>
<tr>
<td>OCP structure</td>
<td>qpDUNES, [ASIPM]</td>
<td>HPMPC, HPIPM, [ASIPM], [FORCES]</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Overview: QP solver types and their way to handle sparsity.  
**underline:** available in acados + support in Simulink  
**gray:** not interfaced in acados, [proprietary]

**efficient condensing from HPIPM:**

- condensing: OCP structured $\rightarrow$ dense, expand solution
- partial condensing: OCP structured with horizon $N$ $\rightarrow$ OCP structured with horizon $N_2 < N$, expand solution, $N_2 \hat{=} \text{qp\_solver\_cond\_N}$
Integration methods in acados

- solve Initial Value Problems (IVP) for
  - Ordinary Differential Equations (ODE)
  - Differential-Algebraic Equations (DAE)
  - + sensitivity propagation (derivative of result with respect to initial state, control input)

- integrators in Python, are ’ERK’, ’IRK’, ’IRK_GNSF’

- size of Butcher table: simp_method_num_stages

- time step is divided into simp_method_num_steps intervals, use the integration method on each interval

- ERK: explicit Runge-Kutta
  - integration order simp_method_num_stages = 1, 2, 4

- IRK: implicit Runge-Kutta
  - Gauss-Legendre Butcher tableaus
    - integration order 2\cdot simp_method_num_stages, A-stable, but not L-stable
  - Gauss-Radau IIA
    - integration order is 2\cdot simp_method_num_stages − 1, L-stable

- IRK_GNSF: structure-exploiting implicit Runge-Kutta method
  - Butcher tableaus as IRK
Python interface

- Continuous time formulation
  - Discretization flexible, cost multiplied with time step, nonuniform grid possible
- Model functions provided as CasADi expressions
- Template workflow
  - Model function and derivatives generated using CasADi
  - C code to set up the OCP solver using the C interface
  - Makefile to compile everything into a shared library
  - Shared library is loaded in Python and used via a wrapper
  - Generated solver can be used in alternative wrapper and embedded framework, C++, ROS
Important Resources

- [https://docs.acados.org/](https://docs.acados.org/)
  - Python API - documents all options in template interface: [https://docs.acados.org/python_api](https://docs.acados.org/python_api)
  - Installation instructions [https://docs.acados.org/installation](https://docs.acados.org/installation)
- Latest acados publication:
  - Mathematical Programming Computation 2021
- [acados forum](https://discourse.acados.org)
- Github examples [https://github.com/acados/acados/tree/master/examples](https://github.com/acados/acados/tree/master/examples)
Problem Shooting – OCP

- check OCP solver status, 0 – Success, other values defined in types.h
- `ocp_solver.print_statistics()`
  - KKT residuals: stat stationarity – Lagrange gradient, eq: equality constraints, ineq: inequality constraints, comp: complementarity
  - `qp_stat`: status of the QP solver, should be 0
  - `qp_iter`: number of iterations in QP solver
- initialization: `set()` – x, u, multipliers
- infeasibility: introduce slacks (soft constraints)
- check failing QP: `get_from_qp_in()`
- try different Hessian approximations: Definiteness & Exactness
  - `ocp.solver_options.hessian_approx = ‘GAUSS_NEWTON’ or ‘EXACT’`
  - Option to turn off exact hessian contributions from cost, constraints or dynamics
  - Set numerical approximation for cost hessian
  - add a Levenberg Marquardt term, `ocp.solver_options.levenberg_marquardt`
- iterates don’t converge:
  - reduce the step size, `ocp_opts.set(‘step_size’, alpha) with alpha < 1`
  - globalization (preliminary implementation) of a merit function based backtracking:
    - `ocp.solver_options.globalization = ‘FIXED_STEP’ or ‘MERIT_BACKTRACKING’`
    - Second order correction: set `globalization_use_SOC` to 1
    - Armijo condition: set `line_search_use_sufficient_descent` to 1.