OPTIMAL CONTROL APPLICATIONS AND METHODS
Optim. Control Appl. Meth. 2011; **32**:298–312
Published online 25 May 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/oca.939

ACADO toolkit—An open-source framework for automatic control and dynamic optimization

Boris Houska*, †, Hans Joachim Ferreau and Moritz Diehl

Electrical Engineering Department (ESAT) and Optimization in Engineering Center (OPTEC), K.U. Leuven, Kasteelpark Arenberg 10, 3001 Leuven, Belgium

SUMMARY

In this paper the software environment and algorithm collection `ACADO Toolkit` is presented, which implements tools for automatic control and dynamic optimization. It provides a general framework for using a great variety of algorithms for direct optimal control, including model predictive control as well as state and parameter estimation. The `ACADO Toolkit` is implemented as a self-contained `C++` code, while the object-oriented design allows for convenient coupling of existing optimization packages and for extending it with user-written optimization routines. We discuss details of the software design of the `ACADO Toolkit 1.0` and describe its main software modules. Along with that we highlight a couple of algorithmic features, in particular its functionality to handle symbolic expressions. The user-friendly syntax of the `ACADO Toolkit` to set up optimization problems is illustrated with two tutorial examples: an optimal control and a parameter estimation problem. Copyright © 2010 John Wiley & Sons, Ltd.

Received 4 September 2009; Revised 10 February 2010; Accepted 10 March 2010

KEY WORDS: dynamic optimization; model predictive control; parameter estimation; optimization software

1. INTRODUCTION

The last decades have seen a rapidly increasing number of applications where control techniques based on dynamic optimization lead to improved performance. These techniques use a mathematical model in form of differential equations of the process to be controlled to predict its future behavior and calculate optimized control actions. This can either be performed once, offline, before the runtime of the process resulting in an open-loop controller. Alternatively, the optimization can be performed, online, during the runtime of the process to obtain a feedback controller. In both cases, the numerical solution of optimal control problems (OCPs) is the main algorithmic step within

*Correspondence to: Boris Houska, Electrical Engineering Department (ESAT) and Optimization in Engineering Center (OPTEC), K.U. Leuven, Kasteelpark Arenberg 10, 3001 Leuven, Belgium.
†E-mail: boris.houska@esat.kuleuven.be

Contract/grant sponsor: Research Council KUL; contract/grant number: CoE EF/05/006
Contract/grant sponsor: Flemish Government; contract/grant numbers: G.0452.04, G.0499.04, G.0211.05, G.0226.06, G.0321.06, G.0302.07, G.0320.08, G.0558.08, G.0557.08, G.0588.09, G.0377.09
Contract/grant sponsor: ICCoS
Contract/grant sponsor: ANMMM
Contract/grant sponsor: MLDM
Contract/grant sponsor: Belgian Federal Science Policy Office; contract/grant number: IUAP P6/04
Contract/grant sponsor: EU
Contract/grant sponsor: Helmholtz-viCERP
Contract/grant sponsor: COMET-ACCM

such advanced controllers. Thus, efficient and reliable optimization algorithms for performing this step—possibly on embedded hardware—are of great interest.

Searching the literature, we can find a number of optimization algorithms which have been implemented for solving OCPs. We can only discuss some of the most common packages: Let us start the list with the open-source package IPOPT [1, 2], originally developed by Andreas Wachter and Larry Biegler, which implements an interior point algorithm for the optimization of large-scale differential algebraic systems. It can be combined with collocation methods for the discretization of the continous dynamic system while a filter strategy is implemented as a globalization technique. IPOPT is written in C/C++ and Fortran, but uses modeling languages such as AMPL or MATLAB in order to provide a user interface and to allow automatic differentiation.

Furthermore, a MATLAB package named PROPT [3] receives more and more attention. PROPT is a commercial tool, developed by the Tomlab Optimization Inc.. PROPT solves optimal control problems based on collocation techniques, while using existing NLP solvers such as KNITRO, CONOPT, SNOPT or CPLEX. Owing to the MATLAB syntax, the package PROPT is more user-friendly than IPOPT—at the price that it is not open-source.

Recently, another open-source code has been published by Brian C. Fabien [4] under the name dsoa. This package is written in C/C++ and discretizes differential algebraic systems based on implicit Runge–Kutta methods. Unfortunately, the package does only implement single-shooting methods, which is often not advisable for nonlinear OCPs. On the optimization level, sequential quadratic programming techniques are employed.

Similar to dsoa, the proprietary package MUSCOD-II, originally developed by Daniel Leineweber [5], is suitable for solving OCPs. MUSCOD-II discretizes the differential algebraic systems based on backward differentiation formula (BDF) or Runge Kutta integration methods and uses Bock's direct multiple shooting [6]. Sequential quadratic programming is used for solving the resulting NLPs. The algorithms implemented in MUSCOD-II are written in C/C++ and Fortran and seem more elaborated than those of dsoa—in particular, because multiple shooting is used instead of single shooting. While MUSCOD-II implements highly efficient code, its software design makes it difficult to extend the code.

Finally, software packages dedicated to nonlinear model predictive control (MPC) in the process industry exist, such as OptCon [7] or NEWCON [8], which are both based on multiple shooting.

Each of the above packages has its particular strengths and all of them have proven successful for a specific range of applications. As they are all tailored to a certain choice of underlying numerical algorithms, it is usually problem dependent which one is most suited. Moreover, their specialized software design renders it difficult to combine algorithmic ideas from different packages or to extend them with new mathematical concepts.

To overcome these issues, the ACADO Toolkit has been designed to meet the following four key properties that are, besides the necessary functionality and efficiency of an implementation, in the authors' opinion crucial for software packages for automatic control based on dynamic optimization. While discussing them, we also sketch how they are addressed within the ACADO Toolkit:

- *Open-source*: The package needs to be freely available at least to academic users for allowing researchers to reproduce all results, to check whether everything is implemented as stated and to try out own modifications.
  The ACADO Toolkit is distributed under the GNU Lesser General Public Licence (LGPL), which even allows the package to be linked against proprietary software. ACADO Toolkit 1.0 is freely available at http://www.acadotoolkit.org.
- *User-friendliness*: The syntax to formulate OCPs should be as intuitive as possible. For experienced users this might only be a question of convenience. However, given the fact that dynamic optimization is more and more widely used in many different engineering applications, also non-experts should be able to formulate their control problems within a reasonable period of time. As far as possible, the software should also automatically make consistent default choices for algorithmic settings and initializations in case they are not provided by the user.

DOI: 10.1002/oca

The ACADO Toolkit makes intensive use of the object-oriented capabilities of C++, in particular operator overloading, which allows to state OCPs in a way that is very close to the usual mathematical syntax. This makes it intuitive to setup optimization problems, even for users who have little programming experience in C/C++.

- *Code extensibility*: The software design should allow to extend the package meeting the following requirements: first, it should be easy to link existing algorithms and second, the design should serve as a platform for new developments while avoiding the duplication of code. To a certain extent, this can be achieved by modularity as long as the efficiency is not affected.
  The ACADO Toolkit realizes these requirements by a careful design of interfaces guaranteeing that almost all algorithmic parts can also be used in their well-documented stand-alone versions. Here, well-established object-oriented software design concepts such as abstract base classes and inheritance are applied.
- *Self-containedness*: The software must only depend on external packages if this is really inevitable; usage of external packages should be optional and the main package should provide a mode to run stand-alone. This feature is particularly crucial for applications on embedded hardware as they usually occur in model predictive controllers: On the one hand, it might even not be possible to link against these packages, e.g. because they rely on different compilers that are not available for the given hardware or simply because linking of external packages is not supported. On the other hand, even if linking is possible, including typically large external packages[‡] significantly increases the size of the executable and the software maintenance effort. Finally, when combining different packages, non-trivial software license issues may arise.
  The ACADO Toolkit is written in a completely self-contained manner. Optionally, external packages for graphical output or specialized linear algebra operations (e.g. sparse solvers for linear systems) can be used.

The paper is organized as follows: Section 2 briefly describes the classes of optimization problems to which the current version 1.0 of ACADO Toolkit can be applied. Afterwards, Section 3 discusses in detail the algorithmic features of ACADO Toolkit to tackle these problem classes. Design and purpose of the main underlying software modules are motivated and described. Two tutorial examples illustrating the use of ACADO Toolkit's syntax are given in Section 4. Section 5 concludes and lists further algorithmic features of ACADO Toolkit that are currently under development and will become part of future releases.

2. PROBLEM CLASSES CONSTITUTING THE SCOPE OF THE SOFTWARE

The ACADO Toolkit 1.0 highlights three important problem classes. The first problem class are offline dynamic optimization problems, where the aim is to find an open-loop control which minimizes a given objective functional. The second class are parameter and state estimation problems, where parameters or unknown control inputs should be identified by measuring an output of a given nonlinear dynamic system. The third class are combined online estimation and MPC problems, where parameterized dynamic optimization problems have to be solved repeatedly to obtain a dynamic feedback control law.

2.1. Optimal control problems

One of the basic problem classes which can be solved with the ACADO Toolkit are standard optimal control problems. These problems typically consist of a dynamic system with differential states $x : \mathbb{R} \to \mathbb{R}^{n_x}$, an optional time-varying control input $u : \mathbb{R} \to \mathbb{R}^{n_u}$ and time constant parameters $p \in \mathbb{R}^{n_p}$. In some cases the formulation of the dynamic system requires also algebraic states, which

[‡]For example, version 3 of the linear algebra library LAPACK has a compiled size of almost 4 MB.

DOI: 10.1002/oca

A general optimal control problem formulation (OCP):

$$
\begin{array}{lrcl}
\underset{x(\cdot),z(\cdot),u(\cdot),p,T}{\text{minimize}} & & \Phi[x(\cdot),z(\cdot),u(\cdot),p,T] & \\
\text{subject to:} & & & \\
\forall t\in[t_0,T]: & 0 & = & f(t,\dot{x}(t),x(t),z(t),u(t),p,T) \\
& 0 & = & r(x(0),z(0),x(T),z(T),p,T) \\
\forall t\in[t_0,T]: & 0 & \geqslant & s(t,x(t),z(t),u(t),p,T)
\end{array}
\qquad \text{(OCP)}
$$

Example for an optimal control problem formulation implemented with ACADO:

```
#include <acado_toolkit.hpp>

int main( ){

    DifferentialState          x;              // a differential state
    AlgebraicState             z;              // an algebraic state
    Control                    u;              // a control
    Parameter                  p;              // a parameter
    DifferentialEquation       f;              // a differential equation

    f << dot(x) == -0.5*x-z+u*u;               // example for a differential-
    f <<      0 ==  z+exp(z)+x-1.0+u;          // algebraic equation.

    OCP ocp( 0.0, 4.0 );                       // OCP with t_0 = 0.0 and T = 4.0
    ocp.minimizeMayerTerm( x*x + p*p );        // a Mayer term to be minimized

    ocp.subjectTo( f );                        // OCP should regard the DAE
    ocp.subjectTo( AT_START, x     == 1.0 );   // an initial value constraint
    ocp.subjectTo( AT_END  , x + p == 1.0 );   // an end (or terminal) constraint

    ocp.subjectTo( -1.0 <= x*u <= 1.0 );       // a path constraint

    OptimizationAlgorithm algorithm(ocp);      // define an algorithm
    algorithm.solve();                         // to solve the OCP.

    return 0;
}
```

Figure 1. A general mathematical formulation and an ACADO example for an optimal control problem.

we denote by $z:\mathbb{R}\to\mathbb{R}^{n_z}$ within this paper. The standard formulation of an OCP is shown in Figure 1.

For standard OCPs, the objective functional Φ is typically a Bolza functional of the form

$$\Phi[x(\cdot),z(\cdot),u(\cdot),p,T]=\int_{t_0}^{T} L(\tau,x(\tau),z(\tau),u(\tau),p,T)\mathrm{d}\tau+M(x(T),p,T). \tag{1}$$

The algorithms, which are currently implemented in the `ACADO Toolkit` assume that the right-hand side function f is smooth or at least sufficiently often differentiable depending on which specific discretization method is used. Moreover, we assume that the function $\partial f/\partial(\dot{x},z)$ is always regular, i.e. the index of the DAE should be one. The remaining functions, namely the Lagrange term L, the Mayer term M, the boundary constraint function r, as well the path constraint function s, are assumed to be at least twice continously differentiable in all their arguments.

Note that Figure 1 shows the general mathematical formulation and also an ACADO implementation example. This example demonstrates how the natural syntax of the toolkit can be used to implement and solve standard OCPs.

DOI: 10.1002/oca

Note that some parts of the above formulation are from a mathematical point of view redundant: For example, a Mayer term can always be formulated as a Lagrange term and vice versa by introducing slack variables. Also the time horizon T and the constant parameter p could be omitted in the formulation above as they can always be eliminated by introducing auxiliary differential states. However, from a numerical point of view it makes sense to use as much structure as possible, such that the above formulation seems natural. Finally, we mention that OCPs contain standard nonlinear programs (NLPs) by leaving away the constraint '`ocp.subjectTo(f)`'.

2.2. Parameter and state estimation

An important class of OCPs, which requires special attention, are state and parameter estimation problems. This subclass of OCPs has also the form. However, as it will be explained in Section 3, parameter estimation problems with least-square objective terms can be treated with a specialized algorithm known under the name generalized Gauss–Newton method. Thus, in the case of a general parameter estimation problem the objective functional Φ takes the form:

$$\Phi[x(\cdot), z(\cdot), u(\cdot), p, T] = \sum_{i=0}^{N} \| h_i(t_i, x(t_i), z(t_i), u(t_i), p) - \eta_i \|_{S_i}^2 .$$

Here, h is called a measurement function, while $\eta_1, \ldots, \eta_N$ are the measurements taken at the time points $t_1, \ldots, t_N \in [0, T]$. Note that the least-squares term is in this formulation weighted with positive semi-definite weighting matrices $S_1, \ldots, S_N$, which are typically the inverses of the variance covariance matrices associated with the measurement errors. In ACADO the syntax

```
ocp.minimizeLSQ( S, h, eta );
```

can be used to define least-square objectives.

2.3. Model-based feedback control

Model-based feedback control constitutes the third main problem class that can be tackled with the `ACADO Toolkit`. It comprises two kinds of online dynamic optimization problems: the MPC problem of finding optimal control actions to be fed back to the controlled process and the moving horizon estimation (MHE) problem of estimating the current process states using measurements of its outputs. The MPC problem is a special case of an (OCP) for which the objective takes typically the form:

$$\Phi[x(\cdot), z(\cdot), u(\cdot), p, T] = \int_{t_0}^{T} \| y(t, x(t), z(t), u(t), p) - y_{\mathrm{ref}} \|_S^2 + \| y^{\mathrm{end}}(x(T), p) - y_{\mathrm{ref}}^{\mathrm{end}} \|_R^2 .$$

Therein, y_{ref} is a tracking reference for the output function y and $y_{\mathrm{ref}}^{\mathrm{end}}$ a reference for a terminal-weight. The matrices S and R are weighting matrices with appropriate dimensions. In contrast to OCPs, MPC problems are assumed to be formulated on a fixed horizon T and employs the above tracking objective function.

In case not all differential states of the process can be measured directly, an estimate has to be obtained using an online state estimator. This is usually done by one of the many Kalman filter variants or by solving an MHE problem. The MHE problem has basically the same form as a parameter estimation problem. Both, the MPC and the MHE problem are solved repeatedly, during the runtime of the process, to yield a model and optimization-based feedback controller.

Note that throughout this paper, the terms MPC and MHE are used for both the linear-quadratic as well as for the general nonlinear case.

3. SOFTWARE MODULES AND ALGORITHMIC FEATURES

We now discuss details of the software design of the `ACADO Toolkit` and describe its main software modules. Along with that we highlight a couple of algorithmic features, in particular

DOI: 10.1002/oca

the functionality to handle symbolic expressions, that give rise to the ACADO Toolkit's unique capabilities.

3.1. The basic structure of the toolkit

The basic structure of ACADO is outlined in Figure 2. In this figure, the sixth most important base classes are shown. Starting at the bottom of the figure, a lower level interface for elementary operations is provided. Classes such as 'Addition', 'Multiplication', etc. inherit from their base class with the name 'Expression'. This class structure is used to build up function evaluation trees. Note that the functionality of this low-level part of ACADO will be explained in Section 3.2.

On the next main level, the base class 'Function' is introduced. Functions can for example consist of symbolic expression trees, or linked C-code, user-written model specification, etc. However, the main concept of this base class is that higher-level algorithms—e.g. integration routines—do not need to know what happens inside, i.e. they can evaluate or differentiate a function independent of whether a symbolic expression tree or a C-function is evaluated in the background. The functions in ACADO automatically communicate to the higher-level algorithms whether they provide automatic differentiation.

While the base class 'Integrator' is an interface for any kind of integration routines the class 'DynamicDiscretization' organizes the discretization techniques in the context of optimal control techniques. Note that the class 'Integrator' can also be used as a base to interface external integration routines. Again, the class 'DynamicDiscretization' hides the specific mode of discretization in a generic way, i.e. for example an SQP algorithm does not need to know whether a differential equation is discretized by collocation or by a shooting method. Moreover, the class 'DynamicDiscretization' could also be used to interface a PDE discretization tool. Note that this form of modularity is organized in such a way that the efficiency is not affected, i.e. an optimization method can always ask the discretization modules for the details, if additional information needs to be passed via suitable data-structures. The details of this approach are however beyond the scope of this paper.

On the higher-level NLP solvers can be interfaced via the base class 'NLPsolver'. NLP solvers are used in the 'OptimizationAlgorithm' which auto-selects and initializes the algorithmic submodules. Specific implementations of the 'OptimizationAlgorithm' inherit from this base class providing tailored drivers for the selected algorithms. For example, the class 'Real-TimeAlgorithm' inherits from 'OptimizationAlgorithm' and implements drivers for e.g. running an SQP method with real-time iterations. Finally, we mention that all the above classes can also be used stand-alone as demonstrated and explained within the tutorial codes that come with ACADO. In this sense, users and developers can choose at which part and also at which level of abstraction they want to use or extend the ACADO toolkit.

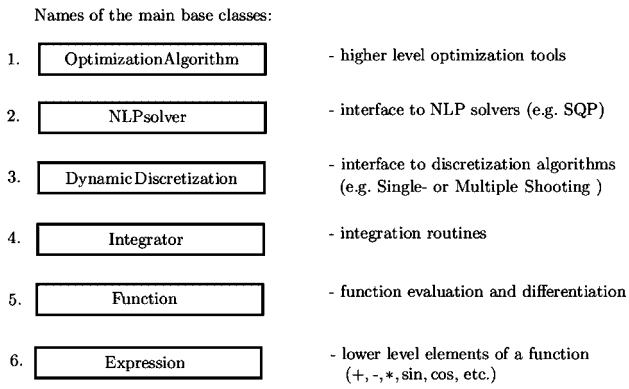


Figure 2. The main algorithmic base classes of ACADO Toolkit.

DOI: 10.1002/oca

Listing 1: Dimension and convexity detection for symbolic functions.

```
int main( ){

    DifferentialState x;
    IntermediateState z;
    TIME              t;
    Function          f;
    z = 0.5*x + 1.0    ;

    f << exp(x) + t    ;
    f << exp(z+exp(z)) ;

    printf("the dimension of f is %d   \n",  f.getDim() );
    printf("f depends on  %d  states   \n",  f.getNX () );
    printf("f depends on  %d  controls \n",  f.getNU () );

    if( f.isConvex() == BT_TRUE )
        printf("all components of function f are convex. \n");

    return 0;
}
```

3.2. *Symbolic expressions*

One of the fundamental requirements for an optimal control package is that functions such as objectives, right-hand sides of differential equations, constraint function, etc. can be provided by the user in a convenient manner. One way to achieve this is that the user links, for example, a simple C function. However, the ACADO Toolkit implements more powerful features: The idea is to use symbolic expressions as a base class to build up complex model equations by making extensive use of the C++ class concept as well as operator overloading. The benefit of this way of implementing functions is that e.g. automatic detection of dependencies and dimensions, automatic as well as symbolic differentiation, convexity detection, etc. are available.

In order to explain this concept, we consider the ACADO tutorial code Listing 1. Compiling and running this simple piece of code with a standard C++ compiler linking ACADO shows—as expected—that the dimension of the defined function f is two and that it depends on one differential state. Moreover, the convexity of the components of f is recognized. Note that these auto-detection routines are typically needed by developers. In most situations, the only remaining work for a user is to define his/her function, whereas the dimension, structure etc. can be detected by the algorithms we want to use. Owing to operator overloading, the syntax can be used as if we would write standard C/C++ code. For example, the intermediate variable z in Listing 1 would only be evaluated once if we evaluate f at a given point, i.e. the symbolic expressions behave as expected.

For a complete overview of the features that are implemented, we refer to the manual [9], where also a lot of commented tutorial codes can be found. In this paper, we can only briefly outline some of the features.

- *Automatic differentiation*: The symbolic notation of functions enables us to provide not only numeric- but also automatic- and symbolic differentiation. The automatic differentiation [10–12] is implemented in its forward as well as in the adjoint mode for first and (mixed) second-order derivatives. Moreover, all expressions can symbolically be differentiated returning again an expression, like AD with source code transformation. This functionality can be used recursively leading to arbitrary orders of symbolic differentiation. Note that the class 'Function' in Figure 2 does not necessarily need to evaluate symbolic expression trees, it is also possible to link a C-function as well as evaluation routines for the corresponding directional derivatives. Thus, it is also possible to link existing AD packages such as ADOL-C [12]. However, using the built-in ACADO routines avoids unnecessary overhead.
- *Convexity detection*: As we have already illustrated in the example code, functions can be tested for convexity/concavity. The corresponding algorithmic routines are based on disciplined convex programming [13]. Note that the syntax for the routines is (almost) the same as in the MATLAB package CVX [14]. As the ACADO code is C++ based, the convexity detection is in

DOI: 10.1002/oca

Listing 2: The definition of a linear function with ACADO.

```
Matrix                    A(3,3);
Vector                    b(3);
DifferentialStateVector   x(3);
Function                  f;

A.setZero();
A(0,0) = 1.0;  A(1,1) = 2.0;  A(2,2) = 3.0;
b(0)   = 1.0;  b(1)   = 1.0;  b(2)   = 1.0;

f << A*x + b;
```

general faster than MATLAB. However, this is a minor advantage in the sense that convexity detection is typically only used as a pre-processing tool.

- *Code optimization*: In the context of optimal control algorithms, right-hand side functions are typically evaluated many times. Thus, it is efficient to pre-optimize functions internally during the initialization phase. In the ACADO Toolkit this pre-optimization is automatically done. For example if a linear function f is defined by the code piece in Listing 2, we would expect that a single evaluation of the function f at a given vector $x \in \mathbb{R}^3$ would involve 12 flops: 9 multiplications and 3 additions, as the matrix–vector product A*x with the matrix $A \in \mathbb{R}^{3\times 3}$ together with the addition of the vector $b \in \mathbb{R}^3$ requires this complexity. However, the ACADO Toolkit auto-detects the zero entries in the matrix A, which in this example is diagonal, such that the evaluation of f costs only six flops—three multiplications and three additions. The price that we have to pay for this internal code optimization is that the 'loading' of the functions takes longer, which is however usually a worthwhile investment of computation time, if f is evaluated very often. Finally, it remains to be mentioned that the ACADO Toolkit would in this case also detect that f is linear.
- *C code generation*: Writing a model function within the ACADO notation does not mean to go into a one way street. A symbolic function can later of course also be exported the in form of (optimized) standard C code.

3.3. Integration algorithms

For the optimization of dynamic systems based on single or multiple shooting methods [6], it is necessary to simulate differential or differential-algebraic equations. In addition, sensitivities of the state trajectory with respect to initial values, control inputs, etc. must be provided. For this aim, ACADO Toolkit comes along with state-of-the-art integration routines such as several Runge–Kutta methods as well as a BDF method which is used for stiff differential or differential algebraic equations. Note that the ACADO BDF integrator, which is based on the algorithmic ideas in [15–17], can also deal with fully implicit differential algebraic equations of index 1, which have the form:

$$\forall t \in [0, T]: \quad F(\dot{y}(t), y(t), u(t), p, T) = 0. \tag{2}$$

Here, differential and algebraic states are merged into one state vector y, whereas u, p, and T are defined as in Section 2.1. However, note that in most OCPs arising in practice the right-hand side F is linear in $\dot{y}$. Moreover, the ACADO BDF integrator uses a diagonal implicit Runge–Kutta starter in order to avoid too small steps taken by the multistep method at the beginning of each multiple shooting interval.

All integrators provide first- and second-order differentiation techniques in order to compute sensitivities of the state trajectory with respect to initial values and control/parameter inputs. Here, the differentiation can either be based on internal numerical differentiation [18, 16] or on (internal) automatic differentiation. However, for automatic differentiation, the right-hand side functions must be provided in the ACADO syntax, i.e. in form of the class Function. For the case that plain C++ or MATLAB functions are linked the expression for the Jacobian should be also provided. Otherwise, numeric differentiation will be used.

DOI: 10.1002/oca

Finally, it should be mentioned that the integration routines that are currently implemented within the ACADO Toolkit are very similar to the existing integrator packages such as Sundials [19] or DAESOL [16] with respect to both the algorithmic strategies as well as the performance. In order to provide consistent and self-contained C++ code, the integration routines have been implemented in cooperation with the class Function, which detects for example the sparsity patterns of the right-hand side functions. However, note that in the current release, the ACADO Toolkit does not only provide dense linear algebra routines, but also provides interfaces for external sparse linear algebra solvers, which can be linked and then be used e.g. within the BDF integrator. Tailored sparse linear algebra solvers will be made available in future versions.

Note that also a stand-alone sub-package ACADO Integrators is available [20], which also provides an elaborate MATLAB interface.

3.4. Discretization of dynamic systems

Once an integrator for dynamic systems is available, the original continous OCP can be discretized. Here, several strategies can be applied. The most simple strategy is to regard the simulation of the system as a function evaluation depending on the initial values, parameters, control inputs, etc. The corresponding discretization method is known under the name single shooting. In the ACADO Toolkit not only single shooting but also multiple shooting methods are implemented, which have turned out to out-perform single shooting methods in many cases [6, 21]. In multiple shooting methods, the whole time interval is divided into several multiple shooting intervals on each of which the dynamic system is discretized using an integrator.

As an alternative to multiple shooting, collocation methods have attracted a lot of attention during the last decades [2, 22]. Here, the dynamic system is discretized at the level of the NLP leading to quite large and sparse NLPs. In the ACADO Toolkit, collocation methods are actually under development and will be released in the near future.

3.5. Nonlinear optimization algorithms

Once a dynamic system can be discretized, the OCPs that have been introduced in Section 2 can be transformed into NLPs. The mathematical standard form of such NLPs is

$$\begin{aligned} &\underset{x}{\text{minimize}} \ \Phi(x), \\ \text{subject to} \quad &G(x)=0, \\ &H(x)\leqslant 0. \end{aligned} \tag{3}$$

Note that in the optimal control context, the discretized NLP has a certain structure. For the case that multiple shooting is used for the discretization, the ACADO Toolkit exploits the structure via condensing techniques that are based on the ideas in [6, 21]. In order to solve the usually nonlinear NLPs, state-of-the-art optimization algorithms are needed. Currently, the ACADO Toolkit provides several SQP-type methods that can e.g. be based on BFGS Hessian approximations, as described in [23], or on Gauss–Newton methods [24]. In addition, line search globalization routines [23, 25] as well as auto-initialization techniques are implemented to make the optimization routines as reliable as possible. In case an underlying quadratic program (QP) becomes infeasible during the SQP iterations, all QP constraints are automatically relaxed using slack variables that are ℓ_1-penalized in the objective function. A tutorial code explaining how these optimization tools can be used will be discussed in Section 4. Note that collocation methods combined with interior point techniques, as e.g. described in [22], are not yet supported in the current release of the ACADO Toolkit.

However, although the ACADO Toolkit comes along with its own optimization routines, it is designed to be extended with existing implementations of optimization algorithms. The software design, which makes use of well-established C++ interface concepts such as abstract base classes and inheritance, allows to use ACADO Toolkit as a test and implementation platform for new developments. For example, in the current implementation the plain C++ code qpOASES [26] is

DOI: 10.1002/oca

linked as a default QP solver. Thus, the ACADO Toolkit is not only designed as a high-end tool for solving optimal dynamic optimization and control problems, but also as a framework that can be filled and extented in many ways.

3.6. Real-time iterations

As mentioned in Section 2.3, MPC problems are a special kind of OCPs. In particular, they depend parametrically on the current initial value x_0 of the process. This special property is exploited within the ACADO Toolkit by applying the real-time iteration scheme presented in [27, 28]. It builds on a direct multiple shooting discretization and only performs one SQP-type iteration using a Gauss–Newton Hessian approximation per feedback loop.

The computations in each iteration are divided into a long 'preparation phase', in which the system linearization, possible elimination of algebraic variables and condensing of the linearized subproblem are performed, and a much shorter 'feedback phase' that only solves one condensed QP. This feedback phase can be orders of magnitude shorter than the feedback phase. In the case of a linear process model, the real-time iteration scheme gives the same feedback as a linear MPC controller. Error bounds and closed-loop stability of the scheme have been established for nonlinear MPC with shifted and non-shifted initializations in [29, 30].

4. TUTORIAL EXAMPLES AND NUMERICAL TESTS

In this section we discuss two code examples: the first one implements a simple time optimal control problem, whereas the second one explains how to set up a simple state and parameter estimation problem with ACADO. Tutorials for closed-loop simulations using real-time iterations can be found on the ACADO Toolkit website [20]. Moreover, the efficiency of the ACADO implementation of real-time iterations is demonstrated in [31], where a online simulation of a kite system is discussed. Note that ACADO is currently designed for systems with 10–100 states [31, 32]. For large-scale systems new algorithmic features need to be added or external optimization packages can be linked.

4.1. An introductory optimal control problem

In this section it is explained how to set up a simple OCP using the ACADO Toolkit. The aim of this tutorial is to solve an example problem of the form:

$$
\begin{array}{ll}
\underset{s(\cdot),v(\cdot),m(\cdot),u(\cdot)}{\text{minimize}} & T \\
\text{subject to:} & \\
\forall t \in [0,T]: & \dot{s}(t) = v(t) \\
& \dot{v}(t) = \dfrac{u(t) - 0.2v(t)^2}{m(t)} \\
& \dot{m}(t) = -0.01\,u(t)^2 \\
& s(0) = 0,\ v(0) = 0,\ m(0) = 1 \\
& s(10) = 10,\ v(10) = 0 \\
& -0.1 \leqslant v(t) \leqslant 1.7 \\
& -1.1 \leqslant u(t) \leqslant 1.1 \\
& 5 \leqslant T \leqslant 15
\end{array}
\qquad (4)
$$

DOI: 10.1002/oca

This problem is based on a simple freespace rocket model with three states: the distance s, the velocity v, and the mass m of the rocket. The aim is to fly in minimum time T from $s(0)=0$ to $s(T)=10$, while constraints on the velocity v and the control input u should be satisfied. The rocket starts with velocity $v(0)=0$ and should stop at the end time T, which can be formulated in the form of the constraint $v(T)=0$.

The corresponding ACADO code, which solves the above OCP numerically, can be found in Listing 3. In this example, we do not specify the NLP solver explicitly, but the class OptimizationAlgorithm chooses by default a multiple shooting discretization method with 20 multiple shooting and control intervals in combination with an SQP algorithm. For the integration, a Runge Kutta solver with order 4 and error control order 5 is chosen. Please note that in this code example no initialization is specified. As mentioned in Section 3.5, an auto-initialization routine has been implemented, which works well for OCPs that are either not too nonlinear or convex. Otherwise an initialization can for example be provided in form of a simple txt-file or as a matrix containing an initial guess for the optimal solution.

Listing 3: An implementation of the optimal control problem (4).

```
int main( ){

    DifferentialState       s,v,m      ;    // the differential states
    Control                 u          ;    // the control input u
    Parameter               T          ;    // the time horizon T
    DifferentialEquation    f( 0.0, T );    // the differential equation

//  ------------------------------------
    OCP ocp( 0.0, T );                      // time horizon of the OCP: [0,T]
    ocp.minimizeMayerTerm( T );             // the time T should be optimized

    f << dot(s) == v;                       // an implementation
    f << dot(v) == (u-0.2*v*v)/m;           // of the model equations
    f << dot(m) == -0.01*u*u;               // for the rocket.

    ocp.subjectTo( f                    );  // minimize T s.t. the model,
    ocp.subjectTo( AT_START, s ==  0.0  );  // the initial values for s,
    ocp.subjectTo( AT_START, v ==  0.0  );  // v,
    ocp.subjectTo( AT_START, m ==  1.0  );  // and m,

    ocp.subjectTo( AT_END  , s == 10.0  );  // the terminal constraints for s
    ocp.subjectTo( AT_END  , v ==  0.0  );  // and v,

    ocp.subjectTo( -0.1 <= v <=  1.7    );  // as well as the bounds on v
    ocp.subjectTo( -1.1 <= u <=  1.1    );  // the control input u,
    ocp.subjectTo(  5.0 <= T <= 15.0    );  // and the time horizon T.
//  ------------------------------------

    OptimizationAlgorithm algorithm(ocp);   // the optimization algorithm
    algorithm.solve();                      // solves the problem.

    return 0;
}
```

In order to visualize the results, a user-friendly Gnuplot interface is available, whose use is outlined in the numerous tutorial examples coming along with the ACADO Toolkit. A Gnuplot screenshot together with the output of the iterations taken by the SQP method is shown in Figures 1 and 3.

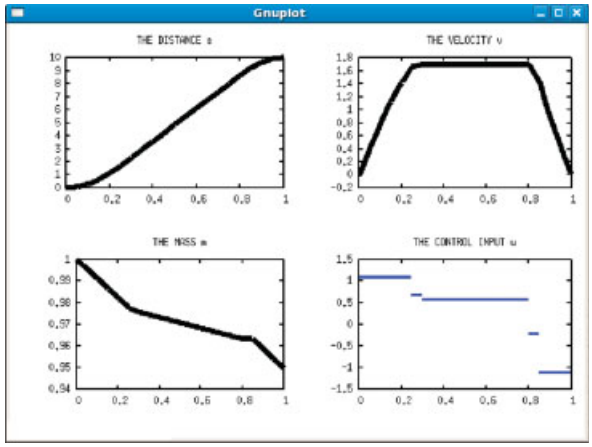
Comparing the implementation in Listing 3 with the corresponding mathematical problem (4) the syntax can quite intuitively be understood. Note that the dimensions of the problem as well as the dependencies have been auto detected. In addition, the structure of the problem is exploited by the numerical algorithm: for example, the control constraints of the form

$$-1.1 \leqslant u(t) \leqslant 1.1$$

are internally detected as bounds. In contrast, e.g. a general constraint of the form

$$u(t)^2 + u(t) \leqslant 1.1$$

DOI: 10.1002/oca



#:	KKT tol.	Obj. Value
1:	1.001e+03	1.000e+01
2:	5.766e+00	9.950e+00
3:	2.946e−02	9.932e+00
4:	7.481e−02	9.906e+00
.	.	.
.	.	.
.	.	.
12:	8.740e−04	7.442e+00
13:	3.308e−07	7.442e+00

convergence achieved.

Figure 3. SQP iteration output and a plot of the optimal results for problem (4).

would have been more expensive as the derivative of the constraint function needs to be evaluated during the SQP iterations. Moreover, the bounds are efficiently used within the QP solver, which is needed during the SQP iterations. Note that all these types of auto-detection routines are a major advance in comparison with most other existing optimal control packages in terms of user-friendliness and automatic generation of efficient code.

4.2. A tutorial parameter estimation problem

Similar to the standard OCP case from the last section, we discuss in this section a tutorial that explains how parameter and state estimation problems can be formulated and solved within the `ACADO Toolkit`. For this aim, we consider the problem

$$\begin{aligned} &\underset{\phi(\cdot),\alpha,l}{\text{minimize}} \quad \sum_{i=1}^{10}(\phi(t_i)-\eta_i)^2 \\ &\text{subject to:} \\ &\forall t\in[0,T]: \quad \ddot{\phi}(t)=-\frac{g}{l}\phi(t)-\alpha\dot{\phi}(t) \\ &\qquad\qquad 0\leqslant\alpha\leqslant 4 \\ &\qquad\qquad 0\leqslant l\leqslant 2 \end{aligned} \tag{5}$$

Here, a simple pendulum model is regarded, which consists of the state ϕ representing the excitation angle; variable $\dot{\phi}$ denotes the angular velocity. The constant $g=9.81$ is the gravitational constant, whereas the friction coefficient α and the length l of the cable are only known to lie between certain bounds. We assume that the state ϕ has been measured at several times.

In Listing 4 a tutorial code is shown which solves problem (5) numerically. Note that the data file, which is read by the routine, is shown in the left part of Figure 4. Here, 2 of the 10 measurements were not successful leading to '`nan`' entries in the data file. Moreover, the measurements have not been taken on a equidistant time grid. Nevertheless, the `ACADO` code, which solves the above parameter estimation problem, is easily set up and deals automatically with the non-equidistant measurements and with the failures in the measurement data.

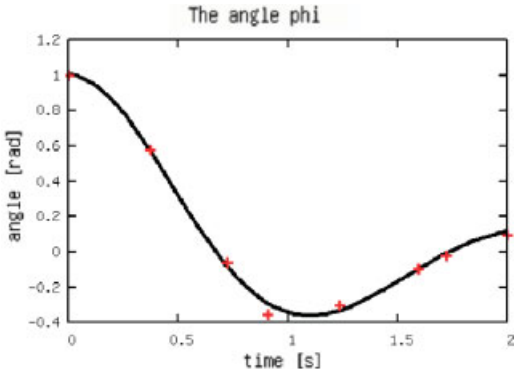
DOI: 10.1002/oca

Listing 4: An implementation of the parameter estimation problem (5).

```
int main( ){

    DifferentialState           phi, dphi;   // the states of the pendulum
    Parameter                   l, alpha ;   // its length and the friction
    const double                g = 9.81 ;   // the gravitational constant
    DifferentialEquation        f        ;   // the model equations
    Function                    h        ;   // the measurement function

//  ------------------------------------------
    OCP ocp( 0.0, 2.0 )                      ;   // construct an OCP
    h << phi                                 ;   // the state phi is measured
    ocp.minimizeLSQ( h, "data.txt" )         ;   // fit h to the data

    f << dot(phi ) == dphi                   ;   // a symbolic implementation
    f << dot(dphi) == -(g/l) * sin( phi )    ;   // of the model
                      -alpha * dphi          ;   // equations

    ocp.subjectTo( f                        );   // solve OCP s.t. the model,
    ocp.subjectTo( 0.0 <= alpha <= 4.0      );   // the bounds on alpha
    ocp.subjectTo( 0.0 <=   l   <= 2.0      );   // and the bounds on l.
//  ------------------------------------------

    ParameterEstimationAlgorithm algorithm(ocp); // the parameter estimation
    algorithm.solve();                           // solves the problem.

    return 0;
}
```

ACSII file "data.txt" containing the measurements:

TIME POINTS	MEASUREMENTS
0.00000e+00	1.00000e+00
2.72321e−01	nan
3.72821e−01	5.75146e−01
7.25752e−01	−5.91794e−02
9.06107e−01	−3.54347e−01
1.23651e+00	−3.03056e−01
1.42619e+00	nan
1.59469e+00	−9.64208e−02
1.72029e+00	−1.97671e−02
2.00000e+00	9.35138e−02




```
The fitting results:
-----------------------------
l     = 1.001e+00 +/- 1.734e-01
alpha = 1.847e+00 +/- 4.059e-01
```

Figure 4. Data file containing the measurements as well as the fitting results obtained by the Gauss–Newton method applied to problem (5).

Note that the parameter estimation algorithm chooses by default a Gauss–Newton SQP method using the structure of the least-squares objective. In the output of the method, the result for the parameter estimation is displayed in the form which is shown in the right part of Figure 2. Note that the computation of the standard deviations of the parameter estimates is based on a linear approximation in the optimal solution as proposed in [24].

5. CONCLUSIONS AND OUTLOOK

The `ACADO Toolkit`, a software environment and algorithm collection for automatic control and dynamic optimization, has been presented. It is an open-source (GNU Lesser Public License) software package written in `C++` that is completely self-contained, i.e. linkage of external packages is optional. Its software design allows to easily extend the `ACADO Toolkit` with the existing numerical optimization packages and its user-friendly syntax makes it very convenient to set up

DOI: 10.1002/oca

customized optimization problems. We briefly sketched the main classes of optimization problems to which it can currently be applied, namely optimal control, state/parameter estimation, MPC, and MHE. Furthermore, its key algorithmic features were outlined, where in particular the functionality to handle symbolic expressions seems to be unique among other optimal control packages.

The `ACADO Toolkit` is currently released in version `1.0` that implements the functionality described in this paper. However, several algorithmic extensions such as collocation techniques, interior point methods, sequential convex programming methods, and multi-objective optimization tools are currently under development which will become part of future releases. Finally, external developers are invited to realize their own algorithmic ideas within the open framework of the `ACADO Toolkit`.

ACKNOWLEDGEMENTS

Research supported by Research Council KUL: CoE EF/05/006 Optimization in Engineering (OPTEC), IOF-SCORES4CHEM, GOA/10/009 (MaNet), GOA/10/11, several Ph.D./postdoc and fellow grants; Flemish Government: FWO: PhD/postdoc grants, projects G.0452.04, G.0499.04, G.0211.05, G.0226.06, G.0321.06, G.0302.07, G.0320.08, G.0558.08, G.0557.08, G.0588.09,G.0377.09, research communities (ICCoS, ANMMM, MLDM); IWT: PhD Grants, Belgian Federal Science Policy Office: IUAP P6/04; EU: ERNSI; FP7-HDMPC, FP7-EMBOCON, Contract Research: AMINAL. Other: Helmholtz-viCERP, COMET-ACCM. The second author holds a PhD fellowship of the Research Foundation—Flanders (FWO).

REFERENCES

1. Wachter A. An interior point algorithm for large-scale nonlinear optimization with applications in process engineering. *Ph.D. Thesis*, Carnegie Mellon University, 2002.
2. Wachter A, Biegler L. IPOPT—an Interior Point OPTimizer, 2009. Available from: https://projects.coin-or.org/Ipopt.
3. PROPT: Matlab Optimal Control Software (ODE,DAE), 2009. Available from: http://tomdyn.com.
4. Fabien BC. dsoa: The implementation of a dynamic system optimization algorithm. *Optimal Control Applications and Methods* 2009; DOI: 10.1002/oca.898.
5. Leineweber DB, Bauer I, Bock HG, Schloder JP. An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part I: theoretical aspects. *Computers and Chemical Engineering* 2003; **27**:157–166.
6. Bock HG, Plitt KJ. A multiple shooting algorithm for direct solution of optimal control problems. *Proceedings 9th IFAC World Congress Budapest*. Pergamon Press: Oxford, 1984; 243–247.
7. Simon LL, Nagy ZK, Hungerbuehler K. Swelling constrained control of an industrial batch reactor using a dedicated NMPC environment: OptCon. *Nonlinear Model Predictive Control*. Lecture Notes in Control and Information Sciences, vol. 384. Springer: Berlin, 2009; 531–539.
8. Romanenko A, Pedrosa N, Leal J, Santos L. Seminario de Aplicaciones Industriales de Control Avanzado. *A Linux Based Nonlinear Model Predictive Control Framework*, Madrid, Spain, 2007; 229–236.
9. Houska B, Ferreau HJ. ACADO Toolkit User's Manual, 2009. Available from: www.acadotoolkit.org.
10. Bischof CH, Carle A, Corliss G, Griewank A, Hovland P. ADIFOR Generating derivative codes from Fortran programs. *Scientific Programming* 1992; **1**:11–29.
11. Griewank A. *Evaluating Derivatives*, *Principles and Techniques of Algorithmic Differentiation.* Frontiers in Applied Mathematics, vol. 19. SIAM: Philadelphia, 2000.
12. Griewank A, Juedes D, Mitev H, Utke J, Vogel O, Walther A. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *Technical Report*, Technical University of Dresden, Institute of Scientific Computing and Institute of Geometry, 1999. Updated version of the paper published in *ACM Transactions on Mathematical Software* 22 1996; 131–167.
13. Grant M, Boyd S. *Graph Implementations for Nonsmooth Convex Programs*, *Recent Advances in Learning and Control*. Lecture Notes in Control and Information Sciences, Springer: Berlin, 2008; 95–110.
14. Grant M, Boyd S. CVX: Matlab software for disciplined convex programming (web page and software), June 2009. Available from: http://stanford.edu/ boyd/cvx.
15. Ascher UM, Petzold LR. *Computer Methods for Ordinary Differential Equations and Differential–Algebraic Equations*. SIAM: Philadelphia, 1998.
16. Bauer I. Numerische Verfahren zur Losung von Anfangswertaufgaben und zur Generierung von ersten und zweiten Ableitungen mit Anwendungen bei Optimierungsaufgaben in Chemie und Verfahrenstechnik. *Ph.D. Thesis*, Universitat Heidelberg, 1999.
17. Petzold LR. DASSL, June 1991. Available from: http:///www.engineering.ucsb.edi/cse/ddassl.tar.gz.
18. Bock HG. *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*, Bonner Mathematische Schriften, vol. 183. Universitat Bonn: Bonn, 1987.

19. Sundials—SUite of Nonlinear and Differential/ALgebraic Equation Solvers (web page and software), 2009. Available from: https://computation.llnl.gov/casc/sundials.
20. ACADO Toolkit Homepage, 2009. Available from: http://www.acadotoolkit.org.
21. Leineweber DB. *Efficient Reduced SQP Methods for the Optimization of Chemical Processes Described by Large Sparse DAE Models*, *Fortschritt-Berichte VDI Reihe 3*, *Verfahrenstechnik*, vol. 613. VDI-Verlag: Dusseldorf, 1999.
22. Biegler LT. An overview of simultaneous strategies for dynamic optimization. *Chemical Engineering and Processing* 2007; **46**:1043–1053.
23. Powell MJD. A fast algorithm for nonlinearly constrained optimization calculations. In *Numerical Analysis*, *Dundee*, 1977, Watson GA (ed.). Lecture Notes in Mathematics, vol. 630. Springer: Berlin, 1978.
24. Bock HG. Recent advances in parameter identification techniques for ODE. In *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, Deuflhard P, Hairer E (eds). Birkhauser: Boston, 1983.
25. Han SP. A globally convergent method for nonlinear programming. *Journal of Optimization Theory and Applications* 1977; **22**:297–310.
26. qpOASES Homepage, 2007–2009. Available from: http://www.qpOASES.org.
27. Diehl M. *Real-Time Optimization for Large Scale Nonlinear Processes*, Fortschr.-Ber. VDI Reihe 8, Me-, Steuerungs- und Regelungstechnik, vol. 920. VDI-Verlag: Dusseldorf, 2002. Available from: http://www.ub.uni-heidelberg.de/archiv/1659/.
28. Diehl M, Bock HG, Schloder JP, Findeisen R, Nagy Z, Allgower F. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control* 2002; **12**:577–585.
29. Diehl M, Findeisen R, Allgower F, Bock HG, Schloder JP. Nominal stability of the real-time iteration scheme for nonlinear model predictive control. *IEE Proceedings—Control Theory and Applications* 2005; **152**(3):296–308.
30. Diehl M, Findeisen R, Allgower F. A stabilizing real-time implementation of nonlinear model predictive control. In *Real-Time and Online PDE-Constrained Optimization*, Biegler L, Ghattas O, Heinkenschloss M, Keyes D, van Bloemen Waanders B (eds). SIAM: Philadelphia, 2007; 23–52.
31. Ferreau HJ, Houska B, Diehl M. Numerical methods for embedded optimisation and their implementation within the ACADO toolkit. In *7th Conference—Computer Methods and Systems* (*CMS'09*), Mitkowski W, Tadeusiewicz R, Ligeza A, Szymkat M (eds). Krakow: Poland, 2009. Oprogramowanie Naukowo-Techniczne.
32. Logist F, Houska B, Diehl M, Impe Jv. Fast pareto set generation for nonlinear optimal control problems with multiple objectives. *Structural and Multidisciplinary Optimization* 2010; DOI: 10.1007/s00158-010-0506-x.