

# Parametric Sensitivities for Optimal Control Problems using Automatic Differentiation

Roland Griesse  
Chair of Mathematics in Engineering  
University of Bayreuth  
95440 Bayreuth  
Germany  
roland.griesse@uni-bayreuth.de

Andrea Walther  
Institute of Scientific Computing  
Technische Universität Dresden  
01062 Dresden  
Germany  
awalther@math.tu-dresden.de

# Parametric Sensitivities for Optimal Control Problems using Automatic Differentiation

## Abstract

This article presents a new area of application for Automatic Differentiation (AD): Computing parametric sensitivities for optimisation problems. For an optimisation problem containing parameters which are not among the optimisation variables, the term parametric sensitivity refers to the derivative of an optimal solution with respect to the parameters. We treat non-linear finite- and infinite-dimensional optimisation problems, in particular optimal control problems involving ordinary differential equations with control and state constraints, and compute their parametric sensitivities using AD. Particular attention is given to the generation of second-order derivatives required in the process.

**Keywords:** Parametric Sensitivity, Optimal Control, Automatic Differentiation

# 1 Introduction

Parametric sensitivity analysis for optimal control problems is a fairly recent discipline. It deals with the computation of an optimal solution's derivatives with respect to parameters which are inherent in the control problem but not optimised for. Let us give a motivating example:

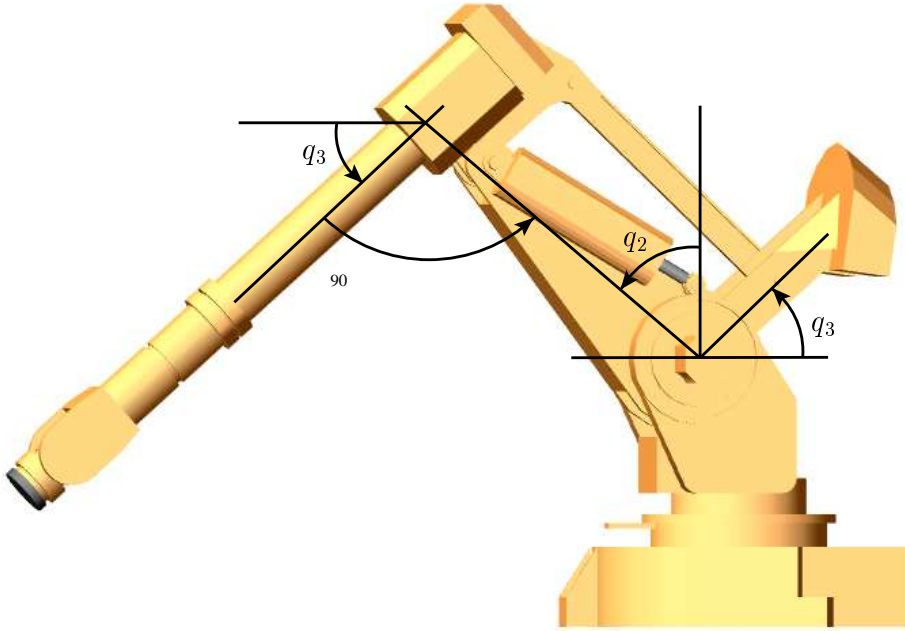


Figure 1: Industrial robot ABB IRB 6400 and its coordinates

An industrial robot—as depicted in Figure 1—is to perform a fast turn-around manoeuvre. Let  $\mathbf{q} = (q_1, q_2, q_3)$  denote the angular coordinates of the robot's joints,  $q_1$  referring to the angle between the base and the two-arm system. The meaning of  $q_2$  and  $q_3$  can be taken from Figure 1. The robot is controlled via three control functions  $u_1$  through  $u_3$ , denoting the respective angular momentum applied to the joints (from bottom to top) by electrical motors.

The control problem under consideration is to minimize the energy-related objective

$$J(\mathbf{q}, \mathbf{u}) = \int_0^{t_f} [u_1(t)^2 + u_2(t)^2 + u_3(t)^2] dt \quad (1)$$

where the final time  $t_f$  is given. The robot's dynamics obeys a system of three

differential equations of second order:

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} = \mathbf{v}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{w}(\mathbf{q}) + \tau_{\text{friction}}(\dot{\mathbf{q}}) + \tau_{\text{reset}}(\mathbf{q}) + \mathbf{u} \quad (2)$$

where  $\mathbf{M}(\mathbf{q})$  is a  $3 \times 3$  symmetric positive definite matrix containing moments of inertia, called a generalized mass matrix. The vector  $\mathbf{v}$  is composed of centrifugal and Coriolis force entries, and  $\mathbf{w}$  contains the gravitational influence. Finally, we allow for forces induced by dry friction and reset forces by means of  $\tau_{\text{friction}}$  and  $\tau_{\text{reset}}$ , respectively. The complete equations of motion can be found in Knauer [21] and Heim [20] and in a forthcoming paper by Büskens and Knauer.

Our robot's task to perform a turn-around manoeuver is expressed by means of initial and terminal conditions:

$$\begin{aligned} q_1(0) &= \pi/2 & \dot{q}_1(0) &= 0 & q_1(t_f) &= -\pi/2 & \dot{q}_1(t_f) &= 0 \\ q_2(0) &= 0 & \dot{q}_2(0) &= 0 & q_2(t_f) &= 0 & \dot{q}_2(t_f) &= 0 \\ q_3(0) &= 0 & \dot{q}_3(0) &= 0 & q_3(t_f) &= 0 & \dot{q}_3(t_f) &= 0. \end{aligned} \quad (3)$$

The choice of control functions is restricted by the presence of control constraints

$$|u_i(t)| \leq 1 \quad \text{for all } t \in [0, t_f] \text{ and } i = 1, 2, 3 \quad (4)$$

as well as state constraints

$$\begin{aligned} -100^\circ/s &\leq \dot{q}_1(t) \leq 100^\circ/s & -70^\circ &\leq q_2(t) \leq 70^\circ \\ -100^\circ/s &\leq \dot{q}_2(t) \leq 100^\circ/s & -28^\circ &\leq q_3(t) \leq 105^\circ \\ -65^\circ &\leq q_2(t) - q_3(t) \leq 65^\circ \end{aligned} \quad (5)$$

again for all  $t \in [0, t_f]$ . In addition to the optimal solution, it is very useful to have parametric sensitivity information at hand which can be employed for a real-time update of the control functions in case that perturbations in the data occur during the robot's operation. This issue will be addressed in Section 2. For our present example, perturbations likely to occur are in the initial position as well as in the weight of the tool in the robot's hand. The first circumstance is taken care of by substituting the first column in (3) by

$$q_1(0) = \pi/2 - p_1 \quad q_2(0) = -p_2 \quad q_3(0) = -p_3. \quad (6)$$

The additional tool weight  $p_4$  enters into the equations of motions (2) in various terms. We refer to Knauer [21] for details. All parameters have a nominal value of zero.

The remainder of the paper is organized as follows: In Section 2, we point to results concerning the existence of parametric sensitivity functions. We also illustrate their usefulness in real-time applications. Section 3 describes the discretisation process transforming the optimal control problem into a finite-dimensional

optimisation problem which can be solved using standard nonlinear programming software. Also, the first application of Automatic Differentiation will surface there. Section 4 addresses the computation of parametric sensitivities for the finite-dimensional optimisation problem obtained from discretisation. In the process, second derivatives are needed whose generation via AD is taken up. We shall present numerical results for the robot control problem along with its parametric sensitivities in Section 5.

Throughout the paper, we denote (partial) derivatives of a vector function  $f(\mathbf{x}, \mathbf{y})$  by  $f_x$ , or, when referring to the derivatives of the  $i$ -th scalar component of  $f$ , by  $\nabla_x f_i$ .

## 2 Parametric Sensitivities in Optimal Control

Throughout this paper, we shall be concerned with optimal control problems of the following type:

**OC(p)** For a given parameter  $\mathbf{p} \in \mathbb{R}^q$ , find a state/control pair

$$(\mathbf{y}_p, \mathbf{u}_p) \text{ in } W^{1,\infty}(0, t_f; \mathbb{R}^n) \times L^\infty(0, t_f; \mathbb{R}^m)$$

$$\text{which minimizes } \varphi(\mathbf{y}(0), \mathbf{y}(t_f), \mathbf{p}) \quad (7)$$

$$\text{subject to } \dot{\mathbf{y}}(t) = f(\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}) \text{ a.e. on } [0, t_f], \quad (8)$$

$$\text{boundary conditions } \mathbf{B}(\mathbf{y}(0), \mathbf{y}(t_f), \mathbf{p}) = 0, \quad (9)$$

$$\text{mixed constraints } \mathbf{C}(\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}) \leq 0 \text{ a.e. on } [0, t_f], \quad (10)$$

$$\text{and state constraints } \mathbf{S}(\mathbf{y}(t), \mathbf{p}) \leq 0 \text{ on } [0, t_f] \quad (11)$$

where  $\varphi : \mathbb{R}^n \times \mathbb{R}^q \rightarrow \mathbb{R}$ ,  $f : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^q \rightarrow \mathbb{R}^n$ ,  $\mathbf{B} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^q \rightarrow \mathbb{R}^b$ ,  $\mathbf{C} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^q \rightarrow \mathbb{R}^c$  and  $\mathbf{S} : \mathbb{R}^n \times \mathbb{R}^q \rightarrow \mathbb{R}^s$  and  $\leq$  is understood componentwise.

Let us point out that after a simple transformation of the equations of motion into a system of first-order differential equations, including one equation to convert the objective (1) from Lagrange form into Mayer form (7), our initial example fits **OC(p)**. The two constraints in (5) involving  $\dot{q}_i$  induce state constraints of first order, while the remaining constraints lead to state constraints of second order [19].

We refer to a given value of the parameter vector  $\mathbf{p}_0$  as the *nominal parameter* and to a corresponding solution  $(\mathbf{y}_0, \mathbf{u}_0)$  of **OC(p<sub>0</sub>)** as the *nominal solution*. As changes in the parameter  $\mathbf{p}$  occur, the optimal solution will deviate from the nominal solution. In real-time applications such as the industrial robot problem,

re-computing the solution for the new parameter  $p$  is often too time-consuming to be an option. However, one can respond to changes in the parameter  $\Delta\mathbf{p} = \mathbf{p} - \mathbf{p}_0$  using a first-order update of the nominal control:

$$\mathbf{u}_p \approx \mathbf{u}_0 + \frac{d\mathbf{u}_0}{d\mathbf{p}} \Delta\mathbf{p}. \quad (12)$$

The term  $d\mathbf{u}_0/d\mathbf{p}$  denotes the derivative of the nominal solution  $\mathbf{u}_0$  with respect to the parameter vector  $\mathbf{p}$ . This so-called *parametric sensitivity differential* for the control component can be computed along with the nominal solution in order to have it available in real-time applications. Likewise, it is possible to predict the new state trajectory in virtue of

$$\mathbf{y}_p \approx \mathbf{y}_0 + \frac{d\mathbf{y}_0}{d\mathbf{p}} \Delta\mathbf{p}. \quad (13)$$

Applications of this and more refined strategies can be found e.g. in Pesch [29] and [30], Büskens [6], Büskens and Maurer [8], and Augustin and Maurer [1].

The existence and further properties of the sensitivity functions

$$\frac{d\mathbf{u}_0}{d\mathbf{p}} : [0, t_f] \rightarrow \mathbb{R}^{m \times q} \qquad \frac{d\mathbf{y}_0}{d\mathbf{p}} : [0, t_f] \rightarrow \mathbb{R}^{n \times q} \quad (14)$$

have been the subject matter of various papers: Malanowski and Maurer treat problems with mixed constraints (10) in [24], with several state constraints (11) of order one in [25], and with a single higher order state constraint in [26].

In our brief review of sensitivity differentials, we follow the survey article of Maurer and Augustin [27]: In addition to some rather technical prerequisites, second order sufficient conditions for  $\mathbf{OC}(\mathbf{p}_0)$  imply that the parametric sensitivities (14) exist. Moreover, one can define the respective differentials for the adjoint variable as well as for the vector of Lagrange multipliers associated to the constraints (10)–(11).

We recall that the nominal solution of  $\mathbf{OC}(\mathbf{p}_0)$ , along with its adjoint variable, satisfies a multi-point boundary value problem (see e.g. Bryson and Ho [4]). It is noteworthy that the sensitivities satisfy a linearization thereof, i.e. a *linear* multi-point boundary value problem, which is also derived in Maurer and Augustin [27]. A prominent feature is that the control, state and adjoint sensitivities can exhibit discontinuities at points where either the nominal control or the nominal adjoint is discontinuous. The nominal solution touching one of the state constraints (11) or entering or exiting a boundary arc for (10) or (11) also triggers discontinuities in the sensitivity differentials, as reflected by the results in Section 5.

Solving the aforementioned boundary value problem numerically as proposed in [27] is one elegant way to obtain the sought-after sensitivity differentials. While

this *indirect method* promises high accuracy and deep insight into the behaviour of the optimal solution, it requires a profound knowledge of optimal control theory in general and, in particular, of the nominal solution's switching structure. In addition, setting up the boundary value problem is not a trivial task.

These difficulties can be circumvented at the expense of accuracy by following a discretise-then-optimize strategy first suggested for the computation of sensitivities by Büskens [5]. This highly workable *direct approach* is outlined in the following two sections. In short, we discretise the nominal problem  $\mathbf{OC}(\mathbf{p}_0)$  and obtain a finite-dimensional optimisation problem. This technique of solving optimal control problems is described for example in [2], [4], [22], [31]. A proof of convergence of the discretized towards the continuous solution can be found in [23] for a particular case. The discretise-then-optimize approach has the advantage that the parametric sensitivities are more accessible for the finite-dimensional than for the infinite-dimensional problem  $\mathbf{OC}(\mathbf{p}_0)$ .

To the authors' knowledge, this is the first time that Automatic Differentiation techniques are used to generate the second order derivatives required in the calculation of sensitivities as set out in Section 4.

### 3 The Discrete Optimisation Problem

In this paper, we apply a direct method for solving the optimal control problem  $\mathbf{OC}(\mathbf{p})$  numerically. The discretisation of the state and the controls yields a finite-dimensional optimisation problem, more specifically a nonlinear programming (NLP) problem. For that purpose, we choose a constant time step  $h$  and the time grid points  $t^j = (j - 1)h$  for  $1 \leq j \leq N$ . Hence, the time interval  $[0, t_f]$  is divided into  $N - 1$  sub-intervals of equal lengths. We approximate the state and the control at the grid points only:

$$\begin{aligned} \mathbf{v}^j & \text{ approximates the state} & \mathbf{y}(t^j), & \mathbf{v} = (\mathbf{v}^1, \dots, \mathbf{v}^N)^T \in \mathbb{R}^{N \cdot n} \\ \mathbf{x}^j & \text{ approximates the control} & \mathbf{u}(t^j), & \mathbf{x} = (\mathbf{x}^1, \dots, \mathbf{x}^N)^T \in \mathbb{R}^{N \cdot m} . \end{aligned}$$

For now we assume that the initial value for  $\mathbf{v}^1 = \mathbf{y}(0)$  is explicitly given, even if it does contain some of the optimisation variables. An ODE integrator  $\psi$  provides the approximation

$$\mathbf{v} = \psi(\mathbf{x}, \mathbf{p}) = (\psi^1(\mathbf{x}, \mathbf{p}), \dots, \psi^N(\mathbf{x}, \mathbf{p}))^T \in \mathbb{R}^{N \cdot n}$$

of the state  $\mathbf{y}$ , dependent on the initial condition and the right-hand side  $f$  of the ordinary differential equation (8). Note that both the initial condition and the right-hand side  $f$  may also depend on the parameter  $\mathbf{p}$ .

There are two alternatives in formulating an appropriate discrete optimal control problem for  $\mathbf{OC}(\mathbf{p})$ . The first approach is called *full discretisation*. Here, the

discretised control as well as the discretised state serve as optimisation variables in the optimisation process. A large number of equality constraints take the state integration given by the integrator  $\psi$  into account. Therefore, the full discretisation yields a very large and sparse Jacobian matrix for the constraints. On the other hand, the calculation of the objective gradient is quite simple. See e.g. Betts [2] as well as von Stryk [32] for Runge-Kutta discretisations and other collocation methods.

The second technique, termed *recursive discretisation*, treats only the control variables as optimisation variables. Hence, for each evaluation of the objective, one has to perform a forward integration of the state equation. Therefore, one has to deal with comparatively few optimisation variables but a non-trivial objective gradient. This second method is suitable for all classical ODE integrators as for example Runge-Kutta schemes. For more information about these two approaches we refer to Büskens and Maurer [7].

In this report, we utilize the recursive discretisation method. Hence, we end up with the following finite-dimensional optimisation problem:

**DOC(p)** For a given parameter  $\mathbf{p} \in \mathbb{R}^q$ , find a state/control pair

$$(\mathbf{v}_p, \mathbf{x}_p) \text{ in } \mathbb{R}^{N \cdot n} \times \mathbb{R}^{N \cdot m}$$

$$\text{which minimizes } F(\mathbf{x}, \mathbf{p}) = \varphi(\psi(\mathbf{x}, \mathbf{p}), \mathbf{p}) \quad (15)$$

$$\text{with boundary conditions } \mathbf{B}(\psi^1(\mathbf{x}, \mathbf{p}), \psi^N(\mathbf{x}, \mathbf{p}), \mathbf{p}) = 0, \quad (16)$$

$$\text{mixed constraints } \mathbf{C}(\psi^i(\mathbf{x}, \mathbf{p}), \mathbf{x}^i, \mathbf{p}) \leq 0, \quad 1 \leq i \leq N, \quad (17)$$

$$\text{and state constraints } \mathbf{S}(\psi^i(\mathbf{x}, \mathbf{p}), \mathbf{p}) \leq 0, \quad 1 \leq i \leq N, \quad (18)$$

where  $\mathbf{v} = \psi(\mathbf{x}, \mathbf{p}) = (\psi^1(\mathbf{x}, \mathbf{p}), \dots, \psi^N(\mathbf{x}, \mathbf{p}))^T$  approximately satisfies the state ODE (8) of problem **OC(p)**. For the numerical results, the integrator  $\psi$  was represented by the classical fourth-order Runge-Kutta scheme with the constant step size  $h$ . This scheme requires the value of the right hand side and thus the value of the control variables at the intermediate times  $t^j + \frac{1}{2}h$ . As we observed a chattering effect in the control sensitivities when using piecewise linear interpolation, we provide the value of  $\mathbf{x}^{j+\frac{1}{2}}$ , to keep the implementation simple, using piecewise constant interpolation  $\mathbf{x}^{j+\frac{1}{2}} = \mathbf{x}^j$ .

In order to solve **DOC(p)**, we use Philip Gill's code NPSOL, which is one of the most renowned SQP solvers [13]. NPSOL is designed to minimize an arbitrary function subject to constraints which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints. The problems to be solved may contain up to a few hundred constraints and variables, depending on the amount of memory available. An augmented Lagrangian merit function ensures convergence from an arbitrary starting point. NPSOL employs three tests of convergence for problems with nonlinear constraints as present in the case of

**DOC(p)**. A sequence of optimisation variables is considered converged when the norm of the search direction vector is small compared to the norm of the current iterate, when the norm of the reduced gradient is small compared to the current objective value, and when the norm of the residuals of constraints in the working set is small enough, see [13]. For the numerical example in Section 5, we set feasibility tolerance = 1e-14 and optimality tolerance = 1e-14.

The user of NPSOL has to provide Fortran-subroutines that define the objective and nonlinear constraint functions as well as optionally the gradient of the objective and the Jacobian of the nonlinear constraints. In case derivatives are not provided by the user, NPSOL falls back on finite difference approximations. We provide complete derivative information using Automatic Differentiation (AD). This well-studied technique generates exact derivatives for the *output variables* of a given computer program with respect to the *input variables*, using the chain rule and other basic differentiation rules [15]. One distinguishes two modes of Automatic Differentiation: The *forward mode* propagates the derivatives simultaneously with the function evaluation. The second one, the *reverse mode* of AD, first evaluates the function to be differentiated. Subsequently, the derivatives of the output variables with respect to all intermediate values are computed using the chain rule and the values obtained during the function evaluation. Hence, one starts computing the derivatives with respect to the last intermediate values and traverses backwards through the evaluation process until the desired derivatives for the input variables are generated.

The complexity results for both modes of AD are based on the operation count  $O_G$  of the underlying vector function  $G$ . Using the forward mode, one computes the product of the Jacobian  $\nabla G$  and a vector  $v$ , i.e.  $\nabla G v$ , for no more than five times  $O_G$ . Using the reverse mode, one evaluates the product of a vector  $u$  and the Jacobian  $\nabla G$ , i.e.  $u^T \nabla G$ , at no more than five times  $O_G$ , see [15]. It is important to realize that this bound for the reverse mode is completely independent of the number of input variables, i.e. the gradient of a scalar-valued function can be obtained at no more than five times  $O_G$ . Furthermore, it follows immediately that the forward mode of AD allows the computation of Jacobians at an operation count of at most five times the number of input variables times  $O_G$ . Conversely, the reverse mode allows the computation of Jacobians for at most five times the number of output variables times  $O_G$ . However, the memory requirement of the basic reverse mode is proportional to the time needed to evaluate the function  $G$  itself.

There are several tools for the application of AD, e.g. ODYSSÉE (INRIA, France, [10]), ADIFOR (Argonne National Laboratory, USA, [3]) and ADOL-C (Technical University Dresden, Germany, [17]). ODYSSÉE provides the scalar forward mode, i.e.  $\nabla G v$ , and the scalar reverse mode, i.e.  $u^T \nabla G$ , for Fortran 77 codes. The results of the scalar forward mode as well as of the vector forward mode, where one multiplies the Jacobian  $\nabla G$  from the right with a matrix  $V$ , i.e.  $\nabla G V$  can be

computed using ADIFOR again for Fortran 77 codes. In contrast to ADIFOR and ODYSÉE, the package ADOL-C is capable of differentiating C and C++ code in the forward and the reverse mode. In order to apply ADOL-C, we transferred our Fortran code into C-code using the package f2c [11].

The objective  $F(\mathbf{x}, \mathbf{p}) = \varphi(\psi(\mathbf{x}, \mathbf{p}), \mathbf{p})$  is a scalar-valued function depending on many optimisation variables, namely the components of the discretized control  $\mathbf{x}$ . By the considerations above, we use the reverse mode of AD to provide the objective gradient

$$\frac{\partial}{\partial \mathbf{x}} F(\psi(\mathbf{x}, \mathbf{p}), \mathbf{p}) = \varphi_y(\psi(\mathbf{x}, \mathbf{p}), \mathbf{p}) \cdot \psi_x(\mathbf{x}, \mathbf{p}) .$$

It is worth noting that the reverse mode approach in fact produces an approximation of the adjoint variable, automatically generating the integration scheme "adjoint" to the Runge-Kutta scheme  $\psi$ . We refer to [14] for more on the link between the reverse mode and the adjoint variable and to Hager [18] for a detailed discussion of adjoint Runge-Kutta schemes.

As an alternative, one may first derive the adjoint of the continuous problem  $\mathbf{OC}(\mathbf{p})$  and subsequently discretise both the primal problem  $\mathbf{OC}(\mathbf{p})$  and the continuous adjoint problem in the optimisation process. The differences between the two approaches and their impact on the optimisation process are studied in [14] for optimal control problems with pure control constraints and are an area of ongoing research. The answer to the question which method is preferable depends on the specific task under consideration.

The decision whether to use the forward or the reverse mode of AD in order to evaluate the Jacobian of the nonlinear constraints (16)–(18) depends on the comparison between the number of constraints and the number of variables  $\mathbf{x}$  because of the complexity estimates described above. We will discuss the choice of the AD-mode for our robot example as well as details concerning the application of the three AD-tools in Section 5.

In contrast to the full discretization approach, if a recursive discretization is used as proposed in this section, one has to differentiate the integrator  $\psi$  in order to obtain both the gradient  $\frac{\partial}{\partial \mathbf{x}} F(\psi(\mathbf{x}, \mathbf{p}), \mathbf{p})$  and the Jacobian of the mixed and pure state constraints (17)–(18). However, the differentiation of the integrator is not required for the Jacobian of pure control constraints. We observe that an explicit ODE integrator  $\mathbf{x} \mapsto \psi(\mathbf{x})$ , e.g. an explicit Runge-Kutta scheme, without step length control, is a  $C^k$  function if the right hand side is  $C^k$ . Hence, differentiability is ensured. In order to generate the derivative information needed, we apply AD. Even if step length control is employed, the pertaining parts of the code can be excluded from the AD procedure: Provided that the step lengths from the forward integration are used also in the differentiated code, and under the assumption that in the last integration step, the final time  $t_f$  is exactly reached,

the correctness of the derivatives is guaranteed [9]. In other words, one obtains the exact discrete derivative of the cost functional with respect to the parameters. This is not obvious, since the time step size may depend on the parameters. Then, AD used as black-box differentiates through the step size computation, which also influences the derivatives of the cost functional. The fact that in the last step the final time  $t_f$  is reached exactly ensures appropriate corrections of the derivatives, such that they yield the correct values [9]. Alternatively, one may exclude the time step computation from the differentiation as proposed above since it does not influence the derivatives at all. More difficulties arise if implicit integrators are applied, involving the iterative solution of a nonlinear system of equations. For such convergent fix point iterations, the convergence of the corresponding derivatives has been proved but lags behind [16], and a modification of the stopping criteria for the iterative solvers becomes mandatory.

## 4 Sensitivities for the Discrete Problem

In the previous section we have pointed out how optimal control problems can be transformed into finite-dimensional optimisation problems. On the other hand, this section stands in its own right for optimisation problems not necessarily originating in the discretisation of problem  $\mathbf{OC}(\mathbf{p})$ .

We consider the standard problem of nonlinear optimisation

$$\mathbf{NLP}(\mathbf{p}) \quad \text{For a given parameter } \mathbf{p} \in \mathbb{R}^q, \text{ find } \mathbf{x} \in \mathbb{R}^N$$

$$\text{which minimizes } F(\mathbf{x}, \mathbf{p}) \tag{19}$$

$$\text{subject to } G_i(\mathbf{x}, \mathbf{p}) = 0 \quad i = 1, \dots, M_e \tag{20}$$

$$\text{and } G_i(\mathbf{x}, \mathbf{p}) \leq 0 \quad i = M_e + 1, \dots, M \tag{21}$$

where  $F : \mathbb{R}^N \times \mathbb{R}^q \rightarrow \mathbb{R}$  and  $G : \mathbb{R}^N \times \mathbb{R}^q \rightarrow \mathbb{R}^M$ .

In preparation for sensitivity analysis, the following notions are useful: The *Lagrange function* associated with problem  $\mathbf{NLP}(\mathbf{p})$  is given by

$$L(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p}) = F(\mathbf{x}, \mathbf{p}) + \sum_{i=1}^M \mu_i G_i(\mathbf{x}, \mathbf{p}). \tag{22}$$

Under differentiability and regularity assumptions superseded by those imposed in Theorem 1, a *KKT point*  $\mathbf{x}$  of  $\mathbf{NLP}(\mathbf{p})$  distinguishes itself by the existence and uniqueness of Lagrange multipliers  $\mu_i$  such that the *Karush-Kuhn-Tucker* (KKT) *conditions* are satisfied:

$$L_x(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p}) = 0 \tag{23}$$

$$G_i(\mathbf{x}, \mathbf{p}) = 0 \quad \text{for } i = 1, \dots, M_e \tag{24}$$

$$\mu_i G_i(\mathbf{x}, \mathbf{p}) = 0, \quad \mu_i \geq 0, \quad G_i(\mathbf{x}, \mathbf{p}) \leq 0 \quad \text{for } i = M_e + 1, \dots, M. \tag{25}$$

For a KKT point  $\mathbf{x}$ , we denote by  $J(\mathbf{x}, \mathbf{p}) = \{i = M_e + 1, \dots, M \mid G_i(\mathbf{x}, \mathbf{p}) = 0\}$  the set of *active inequality constraints* whereas  $J^*(\mathbf{x}, \mathbf{p}) = \{i \in J(\mathbf{x}, \mathbf{p}) \mid \mu_i > 0\}$  contains the indices of all active inequality constraints with strictly positive Lagrange multiplier.

Second order sufficient conditions (SSC)

$$\begin{aligned} \bar{\mathbf{x}}^T L_{xx}(\mathbf{x}, \mathbf{p}) \bar{\mathbf{x}} &> 0 && \text{for all } \bar{\mathbf{x}} \neq 0 \text{ such that} \\ \nabla_x G_i(\mathbf{x}, \mathbf{p}) \bar{\mathbf{x}} &\geq 0 && \text{for all } i \in J(\mathbf{x}, \mathbf{p}) \text{ and} \\ \nabla_x G_i(\mathbf{x}, \mathbf{p}) \bar{\mathbf{x}} &= 0 && \text{for all } i \in J^*(\mathbf{x}, \mathbf{p}) \cup \{1, \dots, M_e\} \end{aligned} \quad (26)$$

guarantee the KKT point  $\mathbf{x}$  to be in fact an (isolated) local optimal solution.

Again, for a given nominal parameter  $\mathbf{p}_0 \in \mathbb{R}^q$ , we denote a nominal solution of  $\mathbf{NLP}(\mathbf{p}_0)$  by  $\mathbf{x}_0$ . We are primarily concerned with the behaviour of the optimal solution under parameter perturbations, as given by the parametric sensitivity matrix

$$\frac{d\mathbf{x}_0}{d\mathbf{p}} \in \mathbb{R}^{N \times q}. \quad (27)$$

As a by-product, we also obtain sensitivities for the Lagrange multipliers  $\bar{\boldsymbol{\mu}}_0$  which belong to the active constraints (the remaining multiplier sensitivities are naturally zero):

$$\frac{d\bar{\boldsymbol{\mu}}_0}{d\mathbf{p}} \in \mathbb{R}^{|J(\mathbf{x}_0, \mathbf{p}_0)| \times q}. \quad (28)$$

Sensitivity results have been known since the 1970s but are not widely prevalent. Therefore, we re-state the main theorem from Fiacco [12], Theorem 3.2.2, which was published in 1976:

**Theorem 1** *Let  $\mathbf{x}_0$  be a local optimal solution to  $\mathbf{OC}(\mathbf{p}_0)$  with Lagrange multiplier  $\boldsymbol{\mu}_0$ . In a neighbourhood of  $(\mathbf{x}_0, \mathbf{p}_0)$ , let  $F$  and  $G$  be twice continuously differentiable with respect to  $\mathbf{x}$ , and let  $G$ ,  $F_x$  and  $G_x$  be once continuously differentiable with respect to  $\mathbf{p}$ . Let second order sufficient conditions hold at  $\mathbf{x}_0$ , and let the active constraints' gradients  $\nabla_x G_i(\mathbf{x}_0, \mathbf{p}_0)$ ,  $i \in J(\mathbf{x}_0, \mathbf{p}_0) \cup \{1, \dots, M_e\}$ , be linearly independent. Finally, let the strict complementarity slackness condition  $J(\mathbf{x}_0, \mathbf{p}_0) = J^*(\mathbf{x}_0, \mathbf{p}_0)$  hold.*

*Then, in a neighbourhood of  $\mathbf{p}_0$ , there exists a unique, once continuously differentiable map  $\mathbf{p} \mapsto (\mathbf{x}(\mathbf{p}), \boldsymbol{\mu}(\mathbf{p}))$  such that  $(\mathbf{x}(\mathbf{p}_0), \boldsymbol{\mu}(\mathbf{p}_0)) = (\mathbf{x}_0, \boldsymbol{\mu}_0)$  and  $\mathbf{x}(\mathbf{p})$  is a KKT point with Lagrange multiplier  $\boldsymbol{\mu}(\mathbf{p})$ , satisfying the second order sufficient conditions, i.e.  $\mathbf{x}(\mathbf{p})$  is an isolated local minimum for  $\mathbf{NLP}(\mathbf{p})$ . In addition, the set of active constraints is unchanged, the strict complementary condition holds, and the active constraints' gradients are linearly independent at  $\mathbf{x}(\mathbf{p})$ .*

**Remark 1** *Differentiability properties are passed from the continuous problem  $\mathbf{OC}(\mathbf{p})$  to its discretisation  $\mathbf{DOC}(\mathbf{p})$  and thus  $\mathbf{NLP}(\mathbf{p})$  through the integrator  $\psi$  which is assumed smooth (e.g. an explicit Runge-Kutta scheme without step size control). Under appropriate assumptions on  $\varphi$ ,  $f$ ,  $\mathbf{B}$ ,  $\mathbf{C}$  and  $\mathbf{S}$ , the differentiability requisites for Theorem 1 are met.*

*The remaining regularity requisites can be conveniently verified numerically since their violation causes the left hand side matrix in (30) to be singular.*

Given the requisites for Theorem 1, it follows from differentiation of (23)–(25) that the sought-after sensitivity differentials

$$\frac{d\mathbf{x}_0}{d\mathbf{p}} = \frac{d\mathbf{x}}{d\mathbf{p}}(\mathbf{p}_0) \qquad \frac{d\bar{\boldsymbol{\mu}}_0}{d\mathbf{p}} = \frac{d\bar{\boldsymbol{\mu}}}{d\mathbf{p}}(\mathbf{p}_0) \qquad (29)$$

can be obtained from the solution of a symmetric system of linear equalities:

$$\begin{pmatrix} L_{xx}(\mathbf{x}_0, \boldsymbol{\mu}_0, \mathbf{p}_0) & \bar{G}_x(\mathbf{x}_0, \mathbf{p}_0)^T \\ \bar{G}_x(\mathbf{x}_0, \mathbf{p}_0) & 0 \end{pmatrix} \begin{pmatrix} \frac{d\mathbf{x}_0}{d\mathbf{p}} \\ \frac{d\bar{\boldsymbol{\mu}}_0}{d\mathbf{p}} \end{pmatrix} = - \begin{pmatrix} L_{xp}(\mathbf{x}_0, \mathbf{p}_0) \\ \bar{G}_p(\mathbf{x}_0, \mathbf{p}_0) \end{pmatrix} \qquad (30)$$

where  $\bar{G}$  refers to  $G$  with all inactive constraints eliminated.

The *KKT matrix* on the left hand side is of the same type as the matrices defining the QP subproblems in an SQP algorithm which generates the solution  $(\mathbf{x}_0, \boldsymbol{\mu}_0)$ , see Nocedal and Wright [28], or, in the case of NPSOL, Gill et al. [13]. However, most of the actual SQP implementations assume that the Hessian of the Lagrangian  $L_{xx}$  is never known exactly. Hence, user-provided second derivatives are in no way required. In fact,  $L_{xx}$  is substituted e.g. by a quasi-Newton update and this substitute *cannot be used* in (30). On the other hand, the exact Jacobian  $G_x$  of the nonlinear constraints is available due to the usage of AD. Since NPSOL returns the Jacobian at the final iterate,  $\bar{G}_x$  is available post-optimally at no additional cost, whereas the matrices  $\bar{G}_p$ ,  $L_{xx}$ , and  $L_{xp}$  are yet missing.

The first order derivative  $\bar{G}_p$  can be generated by AD similarly like  $G_x$  in Section 3. Behind the scenes, one can again distinguish two cases for each scalar constraint  $g$ : If  $g(\mathbf{x}, \mathbf{p})$  emerges from discretisation of a pure control constraint like (4), then  $g_p$  does not involve differentiation of the ODE integrator  $\psi$ . Yet if  $g(\mathbf{x}, \mathbf{p})$  stems from the discretisation of a mixed constraint (10) or pure state constraint (11), then  $g$  can be written as

$$g(\mathbf{x}, \mathbf{p}) = \mathbf{C}(\psi(\mathbf{x}, \mathbf{p}), \mathbf{x}, \mathbf{p}). \qquad (31)$$

Thus the computation of

$$\frac{d}{d\mathbf{p}}g(\mathbf{x}, \mathbf{p}) = C_y(\psi(\mathbf{x}, \mathbf{p}), \mathbf{x}, \mathbf{p}) \cdot \psi_p(\mathbf{x}, \mathbf{p}) + C_p(\psi(\mathbf{x}, \mathbf{p}), \mathbf{x}, \mathbf{p}) \qquad (32)$$

requires the differentiation using AD with respect to the parameter vector  $\mathbf{p}$  of the integrator  $\psi$  and thus of the right hand side  $f$ , including differentiation of the initial values. Dependent upon the length  $q$  of the parameter vector and the number of scalar constraints, it may be advisable to use either the forward or reverse mode of AD to generate  $\overline{G}_p$ . A more advanced user may as well use the decomposition (32) and decide for each of the three Jacobians involved whether to use the forward mode or the reverse mode of AD.

The generation of second order derivatives using Automatic Differentiation is more involved. If an AD-tool is based on operator overloading like ADOL-C one can use the overloaded operations to propagate not only first order derivatives but in fact derivatives of arbitrary order. Therefore, the computation of the complete Hessian of the Lagrange function  $L(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  results in one single call of the Hessian-driver of ADOL-C. Nevertheless, as was mentioned above, the use of ADOL-C required generating C-code out of our Fortran source using f2c. Accomplishing the necessary adjustments in the generated code was about half a day's work.

All currently available AD-tools for Fortran programs, e.g. ADIFOR or ODYSSÉE, are based on source transformation, i.e. a new source file computing the required derivative information is generated. These tools allow the source generation only for first order derivatives. Therefore, we apply Automatic Differentiation a second time to the generated code for first order derivatives in order to obtain a program part computing the required Hessian. Here, the first idea would be to differentiate the program part computing the Lagrange function  $L(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  with respect to  $\mathbf{x}$  using the forward mode by applying ODYSSÉE or ADIFOR. Setting the vector  $\mathbf{v}$  to a unit vector  $\mathbf{e}_i$ , the resulting code segment computes the corresponding component  $\nabla L(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p}) \cdot \mathbf{v} = L_{x_i}(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  of  $L_x(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$ , see Section 3. In a second step one would apply the reverse mode of ODYSSÉE to differentiate this code segment with respect to  $\mathbf{x}$  and  $\mathbf{p}$ , yielding a function that computes  $L_{xx}(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  and  $L_{xp}(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$ , respectively. This approach is disadvantageous because of the following reason: The AD-tool ODYSSÉE provides no vector reverse mode. Hence each row of  $L_{xx}(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  and  $L_{xp}(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  must be computed separately. As an alternative, one can apply the converse ordering, i.e. one first computes  $L_x(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  using the scalar reverse mode of ODYSSÉE and then one utilizes the vector forward mode of ADIFOR in order to evaluate  $L_{xx}(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  and  $L_{xp}(\mathbf{x}, \boldsymbol{\mu}, \mathbf{p})$  within one call to the differentiated code.

We conclude that the generation of second derivatives is possible using the source transformation technique. However, the generation of the corresponding source code is not trivial and involves the application of two different AD-tools because ADIFOR provides the vector forward mode but no reverse mode. Nevertheless, the effort was acceptable once the differentiation order was sorted out.

After generating all required derivative objects either using ADOL-C or the com-

bination ADIFOR/ODYSSÉE, we computed the sensitivities  $\frac{d\mathbf{x}_0}{d\mathbf{p}}$  and  $\frac{d\bar{\boldsymbol{\mu}}_0}{d\mathbf{p}}$  from (30) using the direct solver DSYSV from LAPACK with diagonal pivoting, with both AD-approaches yielding exactly the same results.

As in our recursive discretisation of  $\mathbf{OC}(\mathbf{p}_0)$  the nominal optimisation variables  $\mathbf{x}_0$  represent the nominal discretised control functions (potential free initial values truncated), their sensitivity matrix  $d\mathbf{x}_0/d\mathbf{p}$ , properly ordered, is an approximation of the control sensitivities, evaluated at the points of the time grid.

As the integrator  $\psi$  generates the discretised nominal state trajectory  $\mathbf{v}_0$  from given optimisation variables  $\mathbf{x}_0$ , an approximation of the state sensitivity  $d\mathbf{v}_0/d\mathbf{p}$  on the time grid can be obtained by the chain rule, using the control sensitivity:

$$\left. \frac{d\mathbf{v}_0}{d\mathbf{p}} \right|_{t_i, i=1, \dots, N_t} \simeq \psi_x(\mathbf{x}_0, \mathbf{p}_0) \cdot \frac{d\mathbf{x}_0}{d\mathbf{p}} + \psi_p(\mathbf{x}_0, \mathbf{p}_0). \quad (33)$$

Büsken [5] has developed these and similar formulae for the adjoint sensitivity  $d\boldsymbol{\mu}_0/d\mathbf{p}$ , for the sensitivity of the constraints  $\mathbf{C}$  and  $\mathbf{S}$ , and for first and second order sensitivity derivatives of the objective  $dF/d\mathbf{p}$  and  $d^2F/d\mathbf{p}^2$  in the context of discretised optimal control problems. Alternatively, a summary of these results can be found in Büsken and Maurer [7].

## 5 Results

In this section, we come back to the introductory control problem for the industrial robot with a final time of  $t_f = 2.05$  seconds. The discretisation of the continuous problem has been carried out by the classical fourth-order Runge-Kutta scheme using a time grid with 82 equidistant points. With this choice, the right hand side  $f$  has to be evaluated at points off the time grid which was realized using constant interpolation of the control variables, cf. Section 3. All run-times have been obtained using a current AMD Athlon XP processor at 1600 MHz with 512 MB of RAM.

The nominal solution for  $\mathbf{p}_0 = \vec{0} \in \mathbb{R}^4$  depicted in Figure 2 suggests that the solution to the continuous problem has the structure shown in Figure 3, containing three boundary arcs. For the junction times  $\tau_i$ ,  $i = 1, \dots, 4$ , one obtains the following approximated values  $\tau_1 \approx 0.22$ ,  $\tau_2 \approx 0.26$ ,  $\tau_3 \approx 1.74$ , and  $\tau_4 \approx 1.82$ . The remaining constraints in (4) and (5) never become active. In order to efficiently perform the turn-around maneuver, the robot's arm is retracted in the course of the motion to reduce the moment of inertia with respect to the major  $q_1$  axis. The objective function value for this solution is  $J = 0.8576925$ .

For the optimisation we computed the objective gradient using the reverse mode of AD and the Jacobian of the nonlinear constraints using the forward mode

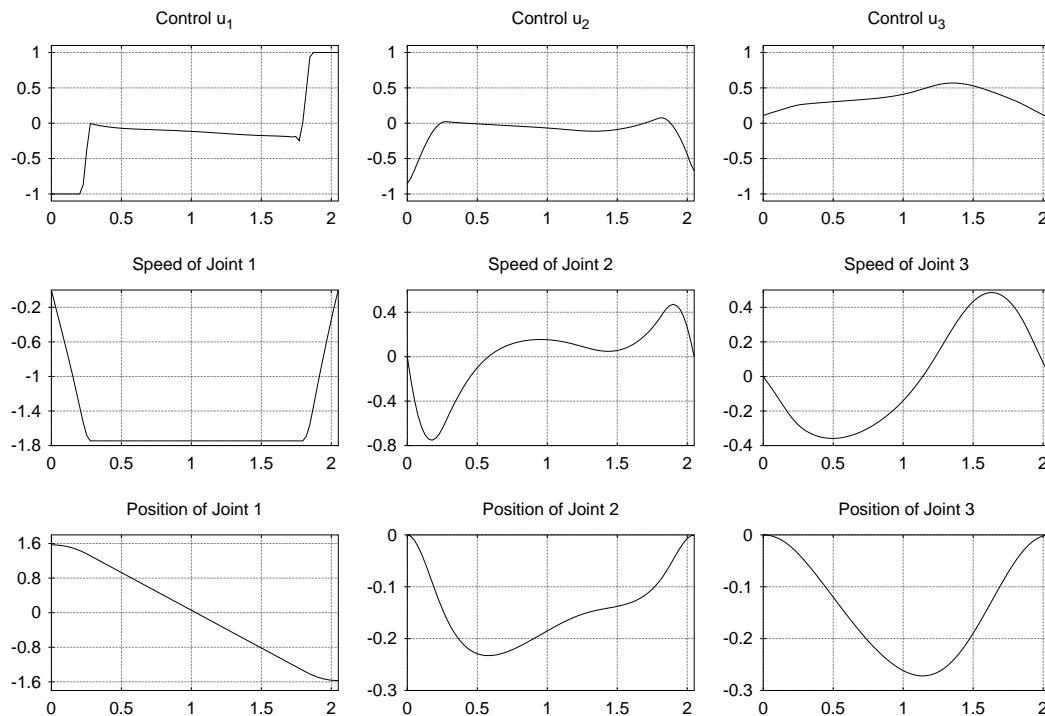


Figure 2: Nominal solution

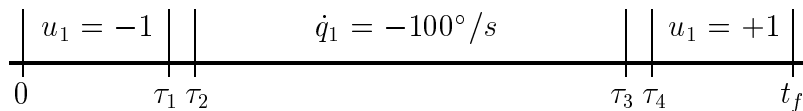


Figure 3: Solution of continuous problem

because there are more constraints than optimisation variables in the robot example. This leads to three possibilities in applying AD, which all yield the same optimal value for  $J$  stated above: First we use ODYSSEE to compute both the gradient of the objective and the Jacobian of the nonlinear constraints. For the latter, we can use only the scalar forward mode, i.e. the Jacobian is evaluated column by column. Secondly, we applied ADOL-C to provide the two derivative objects using the vector forward mode for the Jacobian. Thirdly, we applied ODYSSEE for evaluation of the gradient and ADIFOR for the Jacobian using the vector forward mode. The respective run-times (wall-clock times) of NPSOL for the optimisation process are reported in Table 1. One clearly observes the advantage of the vector mode comparing the run times of the pure ODYSSEE approach to the combination of ODYSSEE and ADIFOR. Furthermore, the run-times illustrate the benefit of the source transformation strategy. Using the technique of operator overloading for implementing AD on one hand one provides flexibility with respect to the differentiation mode and the order of the derivatives. There-

AD-tool(s)	Major iterations	Minor iterations	run-time in $s$
ODYSSÉE	44	68	62.21
ADOL-C	46	67	65.04
ADIFOR/ODYSSÉE	46	72	39.58

Table 1: Run-time details for the optimisation process

fore, the handling is quite simple. On the other hand, one loses the compiler optimisation facilities because of the newly-defined data types required for the overloading approach. This results in run-time penalties.

For the optimisation process, we chose  $\mathbf{x} = \vec{0} \in \mathbb{R}^{3 \cdot 82}$  as initial guess. Using the exact derivatives provided by Automatic Differentiation, NPSOL finds the optimal solution independent of the AD strategy applied. The three AD approaches yield very small deviations of order  $10^{-25}$  in the computed derivatives. These small variations result from differences in the source code computing the derivatives as well as from the two involved programming languages C and Fortran. Despite the fact that the variations in the derivative are so small the resulting iteration count undergoes minor variations as shown in Table 1. Relying on the finite differences strategy implemented in NPSOL, it was impossible to compute even a feasible point starting from the admittedly poor initial guess of the control variables.

After this short report on the optimisation process, we come to the main topic of the paper, i.e. the computation of parametric sensitivities. For the sake of brevity, we report only the sensitivities with respect to parameters  $p_1$  and  $p_4$  which correspond to perturbations in the initial position of joint 1 and in the weight of the robot’s tool, respectively. Already at first glance it is apparent that the optimal solution is far less susceptible to perturbations in the tool’s weight than to changes in the initial position of the same order, see Figures 4 and 5.

According to Section 2, the sensitivity functions  $du_1/d\mathbf{p}$  and  $d\dot{q}_1/d\mathbf{p}$  are expected to have discontinuities at  $\tau_1$  and  $\tau_4$  and at  $\tau_2$  and  $\tau_3$ , respectively. In Figures 4 and 5, these jumps appear as large differences at neighbouring time grid points. Using adaptive mesh refinement, it is possible to resolve the jump conditions and points where they occur to high accuracy, but we do not pursue this aspect here. As expected, the first joint’s sensitivity  $dq_1/dp_1$  is equal to  $-1$  at  $t = 0$  which follows from the initial value condition (6). As  $p_1$  does not enter any other boundary value constraints, all remaining state sensitivities are zero at  $t = 0$  and also at  $t = t_f$ . We observe that the perturbation  $p_1$  has its greatest impact on control component 2 and thus on the state of joint 2.

Note that the sensitivity  $d\dot{q}_1/dp_1$  is zero in the interval  $(\tau_2, \tau_3)$ , where the corresponding constraint  $\dot{q}_1(t) = -100^\circ/s$  is active. The same holds for the control sensitivity  $du_1/dp_1$  on  $(0, \tau_1)$  and  $(\tau_4, t_f)$ , where  $u(t) = \pm 1$  holds. Moreover, this observation is independent of the parameter  $p_i$  under consideration.

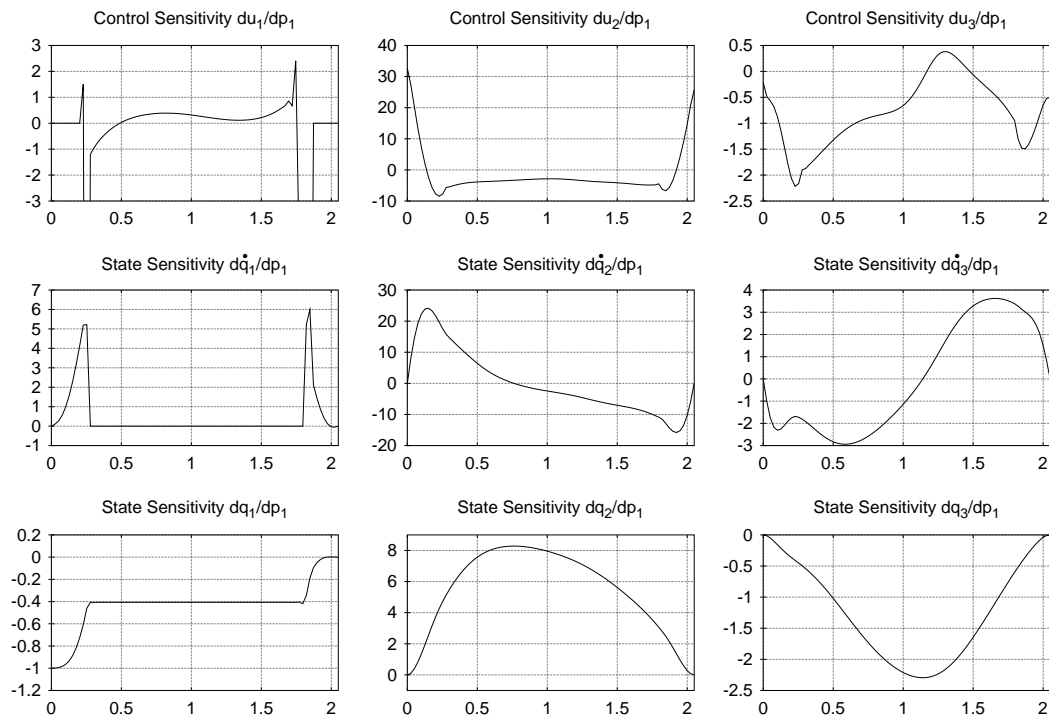


Figure 4: Sensitivities with respect to  $p_1$

AD-tool(s)	sensitivity calculation		overall	
	wall-clock time	user-time	wall-clock time	user-time
ADOL-C	55.31	24.82	120.35	80.35
ADIFOR/ODYSSÉE	8.36	7.79	47.95	45.06

Table 2: Run-time details for the sensitivities and the overall process

The computation of the parametric sensitivities is based on the derivative objects  $\overline{G}_x$ ,  $\overline{G}_p$ ,  $L_{xx}$ , and  $L_{xp}$ , with  $\overline{G}_x$  already available from NPSOL's final iteration. We tested two possibilities to provide  $\overline{G}_p$ ,  $L_{xx}$ , and  $L_{xp}$ . The first one uses ADOL-C while the second applies the ADIFOR/ODYSSÉE combination as described in the preceding section. Table 2 states the corresponding run-times in seconds for the computation of the sensitivities that includes the evaluation of  $\overline{G}_p$ ,  $L_{xx}$ , and  $L_{xp}$  and a solve of the symmetric system (30). Furthermore, Table 2 reports the overall run-times in seconds, i.e. the run-time needed for the optimisation and the calculation of the sensitivities, where only the combination of ADIFOR and ODYSSÉE is considered. It can be inferred that ADOL-C suffers again from the overloading approach, where a lot of memory handling has to be done. This memory traffic accounts for the large gap between user-time (CPU time consumed by a process not counting system calls and memory access) and wall-clock time.

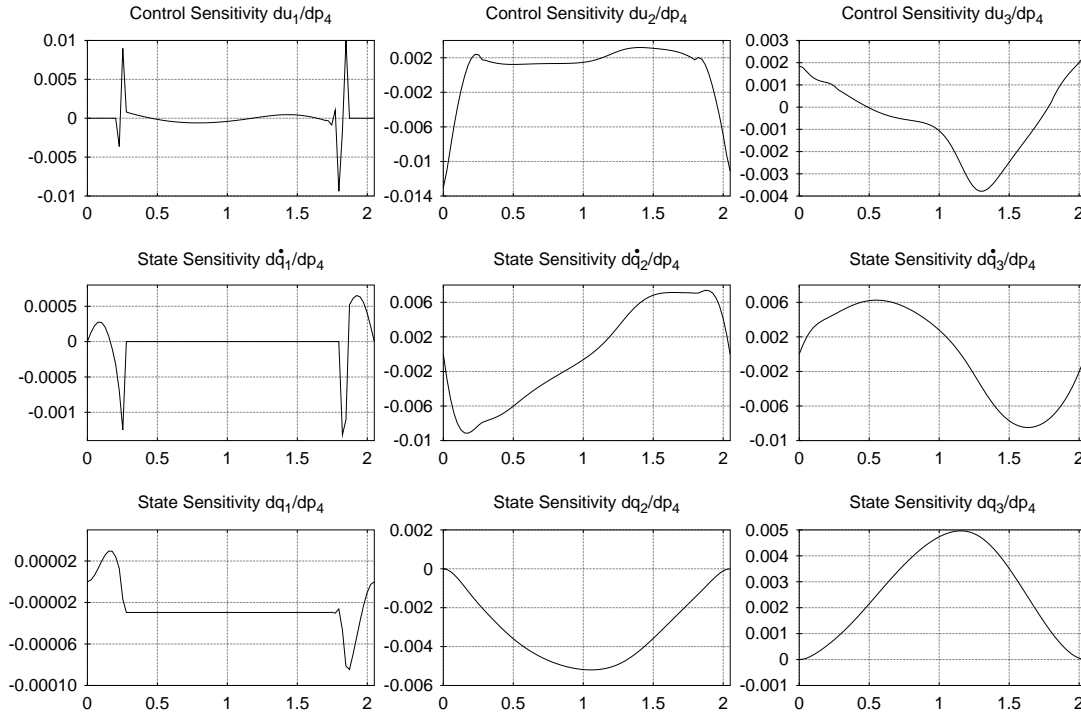


Figure 5: Sensitivities with respect to  $p_4$

Comparing the run-times for the sensitivity calculation from Table 2 with results from a finite difference implementation, it turns out that both Automatic Differentiation approaches were both faster and more accurate. Hence, the usage of Automatic Differentiation is advantageous here, although the computation of the Hessian  $L_{xx} \in \mathbb{R}^{246 \times 246}$  temporarily consumes several hundred megabytes of computer memory. Taking into account that the linchpin of the sensitivities' accuracy is the accuracy of the nominal solution and the derivatives in (30), we advocate for the use of Automatic Differentiation also during the optimization.

In case that one wants to differentiate C-codes instead of Fortran programs, where source transformation AD-tools are not available, the usage of ADOL-C provides the possibility to compute exact derivative and sensitivities considerably fast. Nevertheless, there are many possibilities for a further improvement of ADOL-C which is a current research project at the Technical University Dresden.

## 6 Conclusion

In this paper, we have presented a new area of application for the method of Automatic Differentiation: The computation of parametric sensitivities in nonlinear

constrained optimisation. In particular, we have focused on optimisation problems which arise from the discretisation of optimal control problems for ordinary differential equations.

We have obtained approximations for the optimal control and corresponding optimal state of an industrial robot’s path-planning problem with parameters. Based on the parametric sensitivities for the discretised control problem, we have computed approximations to the parametric sensitivity functions. This direct approach gets by without the intricate procedure of setting up the multi-point boundary value problem for the sensitivity functions.

Particular attention has been given to the generation of second-order derivatives required in the sensitivity computation. For this purpose, we have applied a combination of two AD-tools based on source transformation, namely ADIFOR and ODYSSÉE in order to generate additional code that evaluates the required second-order derivative information. As an alternative, we utilized the AD-tool ADOL-C based on operator overloading. Both approaches provide the required derivative objects to machine precision and—in case of second order derivatives—considerably faster than finite difference techniques.

It is important to recollect that the sensitivities for the finite-dimensional optimisation problem computed from (30) will only be accurate if the KKT matrix and the right hand side can be determined to high precision. This requirement perfectly suits AD techniques as they determine derivatives to machine precision without any numerical error induced e.g. by using finite differences or other derivative approximations. In particular, in order for (30) to yield useful results, it is crucial that the nominal solution  $x_0$  and the Lagrange multipliers  $\mu_0$  be determined accurately by the optimisation routine. The observation that optimisation software usually performs better when provided with exact gradients once more endorses the use of Automatic Differentiation.

## Acknowledgements

The authors are indebted to Christof Büskens for helpful discussions about the topic of parametric sensitivity analysis, and to Matthias Knauer for providing the robot’s equations of motion and images. Furthermore, we would like to thank the referees for their careful reading of the manuscript and suggesting several improvements of the paper.

## References

- [1] Augustin, D., Maurer, H.: *Sensitivity Analysis and Real-Time Control of a Container Crane under State Constraints*, in: Grötschel, Krumke, Rambau: Online Optimization of Large Scale Systems, Springer, 2001.
- [2] Betts, J. T.: *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, Philadelphia, 2001.
- [3] Bischof, C., Carle, A., Khademi, P., Maurer, A., Hovland, P.: *ADIFOR 2.0 User's Guide*, ANL. ANL/MCS-TM-192 (1994).
- [4] Bryson, A., Ho, Y.: *Applied Optimal Control*, Taylor & Francis, 1975.
- [5] Büskens, C.: *Optimierungsmethoden und Sensitivitätsanalyse für optimale Steuerprozesse mit Steuer- und Zustandsbeschränkungen*, Dissertation, Westfälische Wilhelms-Universität Münster, 1998.
- [6] Büskens, C.: *Real-Time Solutions for Perturbed Optimal Control Problems by a Mixed Open- and Closed-Loop Strategy*, in: Grötschel, M., Krumke, S., Rambau, J.: Online Optimization of Large Scale Systems, Springer, 2001
- [7] Büskens, C., Maurer, H.: *Sensitivity Analysis and Real-Time Control of Parametric Optimal Control Problems Using Nonlinear Programming Methods*, in: Grötschel, Krumke, Rambau: Online Optimization of Large Scale Systems, Springer, 2001
- [8] Büskens, C., Maurer, H.: *Nonlinear Programming Methods for Real-Time Control of an Industrial Robot*, Journal of Optimization Theory and Applications, **107**, 505 – 527 (2000)
- [9] Eberhard, P., Bischof, C.: *Automatic Differentiation of Numerical Integration Algorithms*. Mathematics of Computation, **68**, 717 – 731 (1999).
- [10] Faure, C., Papegay, Y.: *Odyssée user's guide*. Version 1.7. Rapport technique 0224, INRIA, September 1998.
- [11] Feldman, S., Gay, D., Maimone, M., Schryer, N.: *A Fortran-to-C converter*. Computing Science Technical Report 149, AT&T Bell Laboratories, 1993.
- [12] Fiacco, A.: *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*, Mathematics in Science and Engineering, Volume 165, Academic Press, New York, 1983.
- [13] Gill, P., Murray, W., Saunders, M., Wright, M.: *User's Guide for NPSOL 5.0: A Fortran Package for Nonlinear Programming*, Technical Report NA 98-2, Department of Mathematics, University of California, San Diego.

- [14] Griesse, R., Walther, A.: *Evaluating Gradients in Optimal Control — Continuous Adjoints versus Automatic Differentiation*, Preprint IOKOMO-11-2001, TU Dresden, submitted.
- [15] Griewank, A.: *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*, Frontiers in Appl. Math. 19, Phil., 2000.
- [16] Griewank, A., Bischof, C., Corliss, G., Carle, A., Williamson, K.: *Derivative convergence for iterative equation solvers*. Optimization Methods and Software, **2**, 321 – 355 (1993).
- [17] Griewank, A., Juedes, D., Utke, J.: *ADOL-C, a package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. Math. Soft., **22**, 131 – 167 (1996).
- [18] Hager, W.: *Runge-Kutta Methods in Optimal Control and the Transformed Adjoint System*, Numer. Math., **87**, 247 – 282 (2000).
- [19] Hartl, R.F., Sethi, S.P., Vickson, R.G.: *A Survey of the Maximum Principles for Optimal Control Problems with State Constraints*, SIAM Review, **37**, 181 – 218 (1995).
- [20] Heim, A.: *Modellierung, Simulation und optimale Bahnplanung von Industrierobotern*, Dissertation, Technische Universität München, 1998.
- [21] Knauer, M.: *Sensitivitätsanalyse verschiedener Gütekriterien bei der optimalen Bahnplanung von Industrierobotern*, Diplomarbeit, Universität Bayreuth, 2001.
- [22] Kraft, D.: *On Converting Optimal Control Problems Into Nonlinear Programming Problems*, Computational mathematical programming, vol. F15 of NATO ASI Series, Springer, 1985.
- [23] Malanowski, K., Büskens, C., Maurer: *Convergence of Approximations to Nonlinear Optimal Control Problems*, in: A. Fiacco: Mathematical Programming with Data Perturbations, Marcel Dekker, 1998
- [24] Malanowski, K., Maurer, H.: *Sensitivity Analysis for Parametric Control Problems with Control-State Constraints*, Computational Optimization and Applications, **5**, 253 – 283 (1996).
- [25] Malanowski, K., Maurer, H.: *Sensitivity Analysis for State Constrained Optimal Control Problems*, Discrete and Continuous Dynamical Systems, **4**, 241 – 272 (1998).

- [26] Malanowski, K., Maurer, H.: *Sensitivity Analysis for Optimal Control Problems Subject to Higher Order State Constraints*, Annals of Operations Research, **101**, 43 – 73 (2001).
- [27] Maurer, H., Augustin, D.: *Sensitivity Analysis and Real-Time Control of Optimal Control Problems*, in: Grötschel, M., Krumke, S., Rambau, J.: *Online Optimization of Large Scale Systems*, Springer, 2001.
- [28] Nocedal, J., Wright, S.: *Numerical Optimization*, Springer Series in Operations Research, Springer, 1999.
- [29] Pesch, H.: *Real-Time Computation of Feedback Controls for Constrained Optimal Control Problems, Part 1: Neighboring Extremals*, Optimal Control Applications & Methods, **10**, 129 – 145 (1989).
- [30] Pesch, H.: *Real-Time Computation of Feedback Controls for Constrained Optimal Control Problems, Part 2: A Correction Method Based on Multiple Shooting*, Optimal Control Applications & Methods, **10**, 147 – 171 (1989).
- [31] von Stryk, O.: *Numerical Solution of Optimal Control Problems by Direct Collocation*, in: Bulirsch, R., Miele, A., Stoer, J., Well, K.H.: *Optimal Control*, International Series of Numerical Mathematics, Birkhäuser, 1993.
- [32] von Stryk, O.: *User's Guide for DIRCOL (Version 2.1): A Direct Collocation Method for the Numerical Solution of Optimal Control Problems*, Fachgebiet Simulation und Systemoptimierung (SIM), Technische Universität Darmstadt, 2000.