URN (Paper): urn:nbn:de:gbv:ilm1-2011iwk-015:6

56TH INTERNATIONAL SCIENTIFIC COLLOQUIUM
Ilmenau University of Technology, 12 Ű 16 September 2011
URN: urn:nbn:gbv:ilm1-2011iwk:5

DIRECT SIMULATION OF MECHANICAL CONTROL SYSTEMS USING ALGORITHMIC DIFFERENTIATION

Klaus Röbenack, Jan Winkler, Carsten Knoll

Institute of Control Theory
Technische Universität Dresden
01062 Dresden, Germany
Email: {klaus.roebenack, jan.winkler, carsten.knoll}@tu-dresden.de

ABSTRACT

A method for direct simulation of mechanical systems is provided which makes use of the algorithmic differentiation package ADOL-C avoiding symbolic or numeric calculation of derivatives. A framework is proposed which makes this software package available under MATLAB/SIMULINK for straightforward simulation purposes. The usefulness of the approach is illustrated by the example of Euler-Lagrange control systems.

Index Terms— Algorithmic differentiation, Lagrangian control system

1. LAGRANGIAN CONTROL SYSTEMS

We consider a mechanical system with n degrees of freedom, which is locally described in the coordinates $\boldsymbol{q} = (q^1, \ldots, q^n)$ on a smooth n-dimensional configuration manifold Q. The system $\boldsymbol{q}$ of generalized coordinates induces a tangent-lifted local coordinate system $(\boldsymbol{q}, \dot{\boldsymbol{q}}) = (q^1, \ldots q^n, \dot{q}^1, \ldots, \dot{q}^n)$ on the tangent bundle TQ. The Lagrangian $L(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a map $L : TQ \to \mathbb{R}$. We treat the external forces $\boldsymbol{u} = (u^1, \ldots, u^n)$ as control or input variables. The Euler-Lagrange equations of a holonomic system are given by

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\frac{\partial}{\partial \dot{\boldsymbol{q}}} L(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \frac{\partial}{\partial \boldsymbol{q}} L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{u} \ . \tag{1}$$

In Eq. (1) we disregarded dissipative forces. Formally, the dissipation of energy can be included in (1) by adding a dissipation term as an appropriate derivative of Rayleigh's dissipation function [1, p. 35]. However, this approach becomes more difficult for sophisticated friction models. For this reason, we add possibly dissipative forces to the external forces, which also simplifies the implementation.

Both for computational purposes and to get better insights into the structure of Eq. (1) we replace the total derivatives in (1) by partial derivatives:

$$\frac{\partial^2}{\partial \dot{\boldsymbol{q}}^2} L(\boldsymbol{q}, \dot{\boldsymbol{q}})\ddot{\boldsymbol{q}} + \frac{\partial^2}{\partial \boldsymbol{q} \partial \dot{\boldsymbol{q}}} L(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \frac{\partial}{\partial \boldsymbol{q}} L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{u} \ . \tag{2}$$

We assume that the Lagrangian L is regular, i.e.,

$$\det \frac{\partial^2}{\partial \dot{\boldsymbol{q}}^2} L(\boldsymbol{q}, \dot{\boldsymbol{q}}) \neq 0 \ .$$

In this case, the second order systems (1) as well as (2) are equivalent to the first order system:

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\boldsymbol{q} = \dot{\boldsymbol{q}} \tag{3a}$$

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\dot{\boldsymbol{q}} = \left(\tfrac{\partial^2}{\partial \dot{\boldsymbol{q}}^2} L(\boldsymbol{q}, \dot{\boldsymbol{q}})\right)^{-1} \cdot \left(\frac{\partial}{\partial \boldsymbol{q}} L(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \frac{\partial^2}{\partial \boldsymbol{q} \partial \dot{\boldsymbol{q}}} L(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{u}\right) . \tag{3b}$$

The right-hand side of (3) defines a control-dependent Lagrangian vector field on TQ, see [2]. System (3) is a special case of the more general form of a nonlinear state-space model

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \ , \tag{4}$$

where the tangent bundle of the configuration manifold TQ is used as the base manifold M of (4), i.e., $M = TQ$ and $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$. For a global description of this system one treats the input-dependent vector field $\boldsymbol{f}$ as a bundle map $\boldsymbol{f} : B \to TM$, where (B, M, π) is a fiber bundle with the total manifold B and the canonical projection $\pi : B \to M$ from the total manifold B to the base manifold M, see [3, 4]. In local coordinates, which will be used for the simulation, the bundle map $\boldsymbol{f}$ has the form $\boldsymbol{f} : \mathbb{R}^{2n} \times \mathbb{R}^m \to \mathbb{R}^{2n}$, i.e., its domain has locally the structure of a cartesian product between state space and input space.

Note that the control input $\boldsymbol{u}$ is not necessarily restricted to external forces as in (1). More generally, one could formulate the mechanical system with a

Lagrangian $L(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$ depending directly on the control $\boldsymbol{u}$, see [5] and [6, Chapter 12]. However, we do not use this approach for our implementation because it is only rarely used among control engineers.

2. ALGORITHMIC DIFFERENTIATION

2.1. Basic principles

In order to obtain (1), (2) or (3) from the Lagrangian L we have to compute first and second order derivatives. Numerical differentiation by divided differences is not well-suited for this task due to truncation and cancellation error. Therefore, the derivatives occurring in these equations are usually computed symbolically with computer algebra packages or libraries, e.g. MATHEMATICA [7], MAPLE [8] or MAXIMA [9].

We suggest the use of an alternative differentiation technique known as *automatic* or *algorithmic differentiation* [10]. Assume that the function $\boldsymbol{F} : \mathbb{R}^N \to \mathbb{R}^M$ under consideration is a sequence of elementary functions and operations. Derivatives of $\boldsymbol{F}$ can be calculated by applying elementary differentiation rules to this sequence. In algorithmic differentiation, all intermediate values are floating point numbers instead of symbolic expressions.

Let the function evaluation of $\boldsymbol{F}$ map a given vector $\boldsymbol{x} \in \mathbb{R}^N$ into $\boldsymbol{z} = \boldsymbol{F}(\boldsymbol{x}) \in \mathbb{R}^N$. In the *forward mode* of algorithmic differentiation, the elementary differentiation rules are applied simultaneuosly to the evaluation of the function values. With this method, a tangent vector $\boldsymbol{v} \in \mathbb{R}^N$ is mapped into the directional derivative

$$\boldsymbol{w} = \boldsymbol{F}'(\boldsymbol{x})\boldsymbol{v} \in \mathbb{R}^M \tag{5}$$

at $\boldsymbol{x}$ in the direction $\boldsymbol{v}$. An example is shown in Tab. 1.

The so-called *reverse mode* of algorithmic differentiation can be interpreted as a generalization of the backpropagation algorithm known from neuronal networks. The differentiation of the elementary functions is carried out in reverse order (compared to the function evaluation). For a given row vector $\bar{\boldsymbol{z}}^T$ with $\bar{\boldsymbol{z}} \in \mathbb{R}^M$ we can calculate the weighted derivative

$$\bar{\boldsymbol{x}}^T = \bar{\boldsymbol{z}}^T \boldsymbol{F}'(\boldsymbol{x}) \tag{6}$$

with $\bar{\boldsymbol{x}} \in \mathbb{R}^N$ in one pass. Therefore, the reverse mode is recommended if $M < N$, especially for $M = 1$, where the full gradient is computed in a single pass.

Carrying out a reverse sweep after a forward pass of algorithmic differentiation allows the efficient calculation of second order derivatives, see [10, Chapter 5] and [11]. In particular, consider $\boldsymbol{F}$ as a function mapping the curve

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \boldsymbol{x}_1 t + \mathcal{O}(t^2) \tag{7}$$

with the Taylor coefficients $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathbb{R}^N$ into a curve

$$\boldsymbol{z}(t) = \boldsymbol{F}(\boldsymbol{x}(t)) = \boldsymbol{z}_0 + \boldsymbol{z}_1 t + \mathcal{O}(t^2) \tag{8}$$

with the coefficients $\boldsymbol{z}_0, \boldsymbol{z}_1 \in \mathbb{R}^M$. The Taylor coefficients

$$\boldsymbol{z}_0 = \boldsymbol{F}(\boldsymbol{x}) \quad \text{and} \quad \boldsymbol{z}_1 = \boldsymbol{F}'(\boldsymbol{x}_0)\,\boldsymbol{x}_1 \tag{9}$$

can be computed directly in the forward mode. Applying the reverse mode with the weighting vector $\bar{\boldsymbol{z}}^T$ yields

$$\mathbf{a}_0^T = \bar{\boldsymbol{z}}^T \tfrac{\partial \boldsymbol{z}_0}{\partial \boldsymbol{x}_0} = \bar{\boldsymbol{z}}^T \boldsymbol{F}'(\boldsymbol{x}_0) \tag{10}$$

$$\mathbf{a}_1^T = \bar{\boldsymbol{z}}^T \tfrac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{x}_0} = \bar{\boldsymbol{z}}^T \boldsymbol{F}''(\boldsymbol{x}_0)\,\boldsymbol{x}_1 \ . \tag{11}$$

Arbitrary second order derivatives can be obtained from (11) by an appropriate choice of the vectors $\boldsymbol{x}_1$ and $\bar{\boldsymbol{z}}$, see [10].

2.2. Derivatives in the Euler-Lagrange equations

For the computation and simulation we treat $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ as independent variables and introduce $\boldsymbol{v} := \dot{\boldsymbol{q}}$. In local coordinates, the Lagrangian L is a map $L : \mathbb{R}^{2n} \to \mathbb{R}$ with the isomorphism $\mathbb{R}^{2n} \cong \mathbb{R}^n \times \mathbb{R}^n$. Having the $2n$-dimensional state-vector

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{v} \end{pmatrix},$$

the gradient of L is given by

$$\frac{\partial L}{\partial \boldsymbol{x}} = \left(\frac{\partial L}{\partial \boldsymbol{q}}, \frac{\partial L}{\partial \boldsymbol{v}} \right). \tag{12}$$

The derivative $\frac{\partial L}{\partial \boldsymbol{q}}$ is required to formulate the equations of motion (2). The second part of the gradient (12), namely

$$\boldsymbol{p} := \frac{\partial L}{\partial \dot{\boldsymbol{q}}} = \frac{\partial L}{\partial \boldsymbol{v}}, \tag{13}$$

consists of the conjugate or generalized momenta occurring in the Legendre transform [2, Section 1.4].

The second order derivative of L is represented by the $(2n \times 2n)$-Hessian matrix

$$\frac{\partial^2 L}{\partial \boldsymbol{x}^2} = \begin{pmatrix} \frac{\partial^2 L}{\partial \boldsymbol{q}^2} & \frac{\partial^2 L}{\partial \boldsymbol{q} \partial \boldsymbol{v}} \\ \frac{\partial^2 L}{\partial \boldsymbol{v} \partial \boldsymbol{q}} & \frac{\partial^2 L}{\partial \boldsymbol{v}^2} \end{pmatrix}. \tag{14}$$

Due to the symmetry of second derivatives we have

$$\frac{\partial^2 L}{\partial \boldsymbol{q} \partial \boldsymbol{v}} = \left(\frac{\partial^2 L}{\partial \boldsymbol{v} \partial \boldsymbol{q}} \right)^T \ .$$

With (12) and (14) we have all derivative required in (2) and (3).

2.3. The principle of operator overloading

The software package ADOL-C used in the following as a variant for implementing algorithmic differentiation makes use of the principle of operator overloading [10]. In this section this principle is discussed in

Function value $F(x,y)$		Derivative values	$\partial F(x,y)/\partial x$	$\partial F(x,y)/\partial y$
x	3.0	$\dot{x}$	1.0	0.0
y	4.0	$\dot{y}$	0.0	1.0
$v_1 := y^2$	16.0	$\dot{v}_1 := 2y\dot{y}$	0.0	8.0
$v_2 := x + v_1$	19.0	$\dot{v}_2 := \dot{x} + \dot{v}_1$	1.0	8.0
$v_3 := \sin(v_2)$	0.149877	$\dot{v}_3 := \dot{v}_2 \cos(v_2)$	0.988705	7.90964
$v_4 := xv_3$	0.449632	$\dot{v}_4 := \dot{x}c_3 + x\dot{v}_3$	3.11599	23.7289
$z := v_4$	0.449632	$\dot{z} := \dot{v}_4$	3.11599	23.7289

Table 1. Simultaneous calculation of the function value and the values of directed derivatives of the example function $z = F(x,y) = x\sin(x+y^2)$ in the forward mode of algorithmic differentiation

more detail. The key idea of operator overloading in algorithmic differentiation is to make use of the fact that every mathematical expression can be decomposed into a sequence of basic mathematical operations like $+$, $-$, $*$, $/$, sin, log, etc. For each of these operations the differentiation rules can be easily applied, e.g. $(xy)' = x'y + xy'$. If one introduces a new C++ class `ddouble` holding not only the value of the variable but also its derivative, e.g.

```
class ddouble
{
 public:
  double val; // function value
  double der; // derivative value
};
```

one is able to utilize operator overloading in order to easily compute the derivatives of each element of an expression and by making use of the chain rule of arbitrary complex expressions as well.

For that we have to provide the usual binary operations for the new class `ddouble`. For example, in case of multiplication we have

```
ddouble operator*(ddouble x,ddouble y)
{
 ddouble z;
 z.val = x.val*y.val;
 z.der = x.val*y.der+y.val*x.der;
}
```

Furthermore, we have to replace all differentiable functions (e.g. sin, cos, exp, log) that act on the type `double` by appropriate methods for the class `ddouble` such that in addition to the function value one also computes the derivative value using elementary differentiation rules in connection with the chain rule. For example, one has

```
ddouble sin (ddouble x)
{
 ddouble z;
 z.val = sin(x.val);
 z.der = x.der*cos(x.val);
}
```

By this approach it is guaranteed that all derivatives are computed using floating point accuracy, i.e., avoiding truncation and cancellation errors.

3. THE SOFTWARE-PACKAGE ADOL-C

In this section we describe the software-package ADOL-C [12, 13] offering one possibility of implementing algorithmic differentiation. It is based on the programming language C/C++ and uses the method of operator overloading. Other variants for implementation of algorithmic differentiation exist like source-code transformation which are not discussed here (e.g., see [10, 14, 15] for more details).

3.1. Preparation of code

In ADOL-C in a first step the code representing e.g. a scalar valued function $F : \mathbb{R}^N \to \mathbb{R}$ is marked as a so called *active section*. Input and output variables are assigned as variables of type `adouble`, similar to the type `ddouble` discussed above. For example, if one has the function

$$z = F(x_1, x_2) = \sqrt{x_1} + x_2^3$$

a possible implementation could look like this:

```
trace_on(tag); // Active section start

// Active variables
adouble ax1,ax2,az;

double x1,x2   // point of expansion
double z;      // function value

// Assignment of independents
ax1 <<= x1;
ax2 <<= x2;

// Evaluation
az = sqrt(x1)+pow(x2,3);

// Assignment of dependents
az >>= z;

trace_off();   // Active section end
```

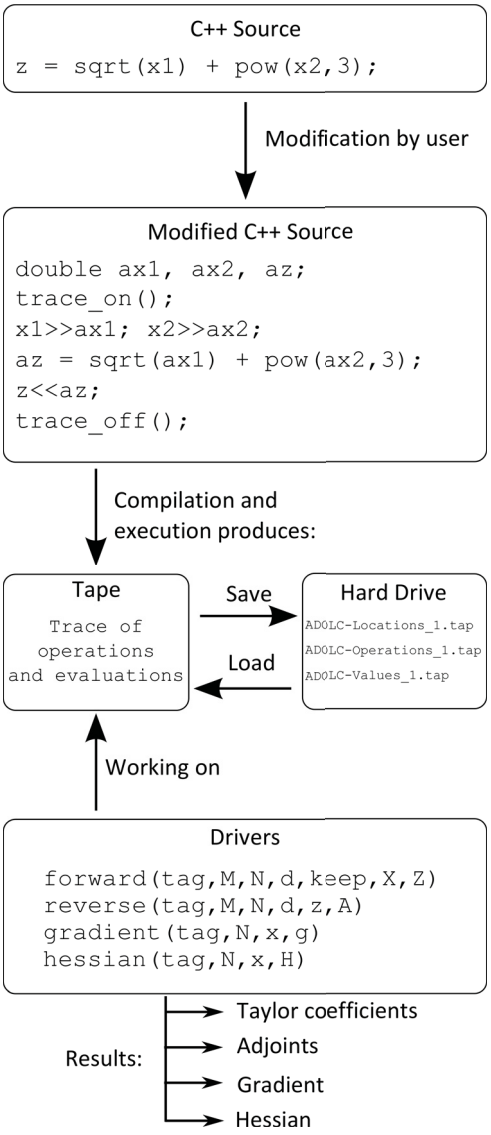


Fig. 1. Workflow in ADOL-C for generating and accessing the tape.

This code creates a data structure called *tape* holding the trace of the function evaluation. This tape is used in the following for the calculation of the several types of derivatives using so called *drivers* which are discussed in the next section. The tape has only to be created once a time for an aribitrary input value x_0. This holds as long as there are no user defined quadratures and all comparisons involving `adouble` variables yield the same result. Thus, repetitive calls of the drivers for different input values differing from the values the tape was generated from may follow. The drivers acting on these tapes provide a C as well as C++ interface, i.e., once a tape is generated it can be used even in environments that do not support C++. The tape is referenced by the integer `tag`.

3.2. ADOL-C Drivers

In this section we describe four of the several drivers that ADOL-C provides: `forward`, `reverse`, `gradient`, and `hessian`. Each of these drivers acts on the tapes generated before.

3.2.1. Drivers for forward and reverse mode

Given a tape of the sufficiently smooth function $\boldsymbol{F} : \mathbb{R}^N \to \mathbb{R}^M$ and the Taylor coefficients $X = (\boldsymbol{x}_k)$ of the series expansion

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \boldsymbol{x}_1 t + \cdots + \boldsymbol{x}_d + \mathcal{O}(t^{d+1}) \qquad (15)$$

of the curve $\boldsymbol{x}$ of the independent variable one can compute the Taylor coefficients $Z = (\boldsymbol{z}_k)$ of the series expansion

$$\boldsymbol{z}(t) = \boldsymbol{F}(\boldsymbol{x}(t)) = \boldsymbol{z}_0 + \boldsymbol{z}_1 t + \cdots + \boldsymbol{z}_d + \mathcal{O}(t^{d+1}) \quad (16)$$

using the ADOL-C function `forward`:

```
int forward(tag,M,N,d,keep,X,Z)
short int tag;    // tape tag of F
int M;            // number of dependent variables
int N;            // number of independent variables
int d;            // highest derivative degree d
int keep;         // flag for reverse mode preparation
double X[N][d+1]; // Taylor coeffs. X = (x_k)
double Z[M][d+1]; // Taylor coeffs. Z = (z_k)
```

In reverse mode, ADOL-C computes the vectors $\mathbf{a_0}, \mathbf{a_1}, \ldots$ as given in (10), (11). They are obtained by calling the ADOL-C function `reverse`:

```
int reverse(tag,M,N,d,z,A)
short int tag;    // tape tag of F
int M;            // number of dependent variables
int N;            // number of independent variables
int d;                 // highest derivative degree d
double z[M];           // weighting vector z
double A[N][d+1]; // resulting A = (a_k)
```

Note that the 2-dimensional arrays are allocated as arrays of pointers.

3.2.2. Gradient

Next to several other drivers, ADOL-C provides some easy-to-use drivers computing the most frequently used derivative objects. The gradient $F'(\boldsymbol{x})$ of a scalar valued function $F : \mathbb{R}^N \to \mathbb{R}$ can be obtained by the driver `gradient`:

```
int gradient(tag,N,x,g)
short int tag; // tape tag of F
int N;         // number of independent variables
double x[N];   // independent vector x
double g[N];   // resulting gradient F'(x)
```

3.2.3. Hessian

If one is interested in the $N \times N$-Hessian matrix $F''(\boldsymbol{x})$ of the function F one may call the driver `hessian`:

```
int hessian(tag,N,x,H)
short int tag;  // tape tag of F
int N;          // number of independent variables
double x[N];    // independent vector x
double H[N][N]; // resulting Hessian matrix F''(x)
```

The full operating principle of ADOL-C is sketched in Figure 1.

4. MATLAB INTERFACE TO ADOL-C

When simulating complex mechanical systems using MATLAB/SIMULINK one may be confronted with the fact that the gradient and the Hessian of the Lagrangian L may be required as discussed in Sections 1 and 2. In order to avoid calculating these elements, e.g. by hand or by a computer algebra system, we offer an interface from MATLAB to ADOL-C. As will be shown it is only required to provide the Lagrangian on its own as a snippet of C++ code while the gradient and Hessian are computed by the corresponding ADOL-C drivers wrapped into Matlab.

MATLAB by itself provides an API to the programming language C. Using this API it is possible to write functions which can be called from MATLAB using its usual syntax. However, these functions are executed as compiled code in contrast to functions written in MATLAB's scripting language which is interpreted. While the first ones are in fact binary libraries (with the file extension `mexw32` under 32-bit MS Windows and `.mexglx` under 32-bit Linux), the latter ones are textfiles with the extension `.m`.

Now the idea is to use this API as a wrapper for the call of the required ADOL-C functions. This is an easy task for wrapping the ADOL-C drivers since one only has to pass the arguments from MATLAB to the ADOL-C functions in an appropriate manner and vice versa. A little bit more complicated is the generation of the tapes from the C++ source code. In order to make this procedure as easy as possible for the user the following procedure is specified by the framework:

1. The user provides the function which is going to be taped as pure C++ code, *without* any headers, preprocessor derictives, etc. For example, if one wants to generate the tape of the function $z = \sqrt{x_1} + x_2^3$, the user just has to provide a file with a single line of valid C++ code:

   ```
   z[0] = sqrt(x[0]) + pow(x[1],3);
   ```

   The only prerequisite is that the independents $x_1, x_2, \ldots$ are named as `x[0]`, `x[1]`, ... The same holds for the dependents z_1, z_2.

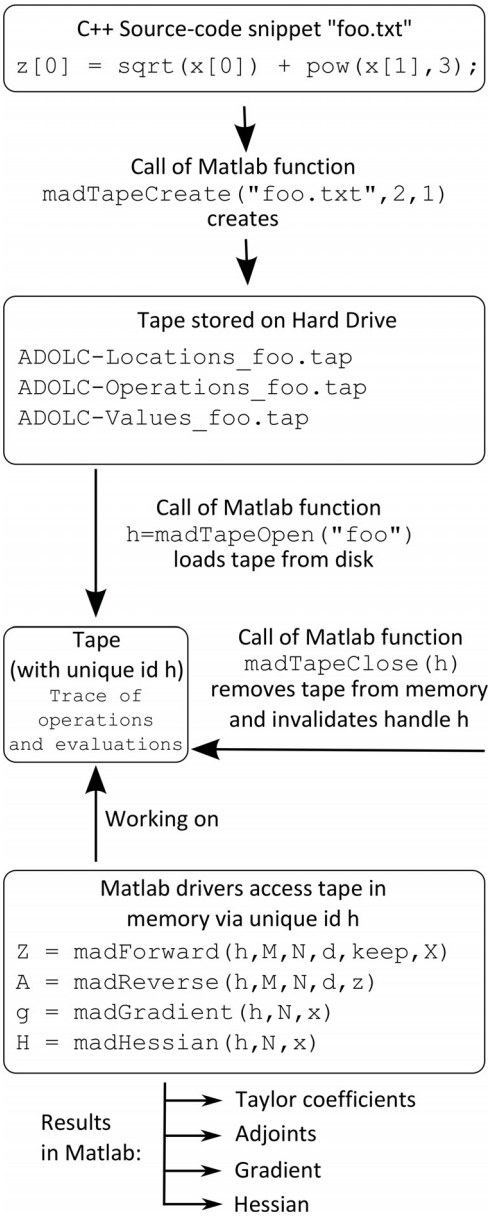
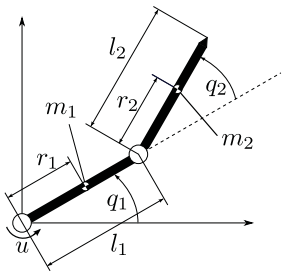


Fig. 2. Workflow under Matlab for generating and accessing the tape.

2. The user has to call the mex-function `madTapeCreate` with appropriate parameters telling MATLAB the number of dependents and independents. Then a script is called building an intermediate C++ file which is than compiled, linked against ADOL-C and executed in order to generate the corresponding tape. This tape is stored on the hard disk using the name of the file which contained the code snippet.

3. By opening the previously generated tape using `madTapeOpen` one receives a handle to the tape which can be used to access the tape using the ADOL-C drivers wrapped into MATLAB. The tape is kept persistently into



m_i ... link masses
l_i ... link lengths
r_i ... distances of the centers of gravity
I_i ... moments of inertia of the links
q_i ... joint angles
u_1 ... torque of active joint

Fig. 3. Underactuated two link manipulator in horizontal plane (top view).

memory between successive calls of the drivers until the function `madClose` is called. The drivers are prefixed by `mad` in MATLAB. Thus, the drivers discussed in Section 3.2 are called `madForward`, `madReverse`, `madGradient`, and `madHessian`.

The workflow of this procedure is sketched in Figure 2. This procedure requires a C/C++ compiler installed on the system. Our wrapper has been successfully tested under Windows using MS Visual C++ 2008/2010 and MinGW/GCC as well as under Linux using GCC.

5. EXAMPLE: UNDERACTUATED MANIPULATOR

In this section we consider the underactuated two-link manipulator in the horizontal plane (cf. Fig. 3) as a simple but famous mechanical benchmark system [16, 17, 18]. As shown in [19] this system violates the so called Brockett condition [3], which means that a single equilibrium point cannot be stabilized by means of a continous time-invariant state feedback. From a perspective of linear time-invariant systems theory this leads to an uncontrolable linearization.

Due to the absence of potential energy the Lagrangian is identical to the kinetic energy

$$L = T = \frac{1}{2}(a_1\dot{q}_1^2 + a_2(\dot{q}_1+\dot{q}_2)^2 + 2a_3\dot{q}_1(\dot{q}_1+\dot{q}_2)\cos q_2) \tag{17}$$

with the parameters $a_1 = I_1 + m_1 r_1^2 + m_2 l_1^2$, $a_2 = I_2 + m_2 r_2^2$ and $a_3 = m_2 l_1 r_2$.

From (3) together with (17) it is obvious that the manipulator system is in rest whenever the angular velocities and the input torque vanish. This means that the set of equilibrium points consists of the whole configuration manifold Q. One of the simplest control strategies is to stabilize not a single equilibrium point but a submanifold Q_1 of Q. In particular, this can be done by applying a simple PD-feedback law on the first joint

$$u_1 = -k_\mathrm{d}\dot{q}_1 - k_\mathrm{p}(q_1 - q_{1,\mathrm{d}}), \quad k_\mathrm{d}, k_\mathrm{p} > 0 \tag{18}$$

with the desired position $q_{1,\mathrm{d}}$. In [19] it is shown that this controller renders the set

$$Q_1 := \{\boldsymbol{q} \in Q : q_1 = q_{1,\mathrm{d}}\} \tag{19}$$

asymptotically stable.

To simulate this control system using the proposed method, in principle one only needs to implement the Lagrangian (17) and the control law (18). However, if there are additional non-conservative generalized forces acting on the joints, they have to be introduced seperately. For example we assume a viscous friction in the second joint, i.e.,

$$u_2 = -d\dot{q}_2 \tag{20}$$

with the coefficient $d > 0$. Given the Lagrangian (17) as a snippet of C++ code and treating $\boldsymbol{q}, \dot{\boldsymbol{q}}$ as independent variables (cf. Section 2.2, then, after having created the corresponding tape using `madTapeCreate`, the functions `madGradient` and `madHessian` can be used for implementation of the model under MATLAB/SIMULINK.

The simulation was carried out with the normalized parameters $a_1 = 0.025$, $a_2 = 0.01$, $a_3 = 0.009$, $k_p = 0.5$, $k_d = 0.2$, $d = 0.002$, the desired position $q_{1,\mathrm{d}}$ and the initial values $\mathbf{q}(0) = (90°, 45°)^T$ and $\dot{\mathbf{q}}(0) = \mathbf{0}$. The simulation results are shown in Fig. 4 and 5.

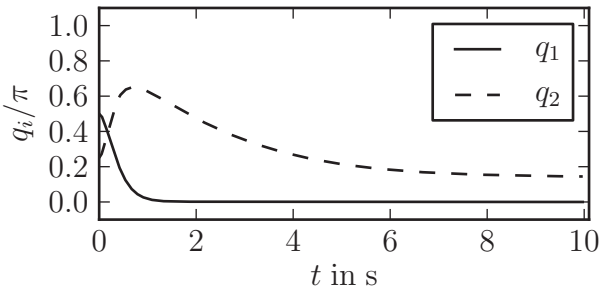


Fig. 4. Angles of the underactuated manipulator.

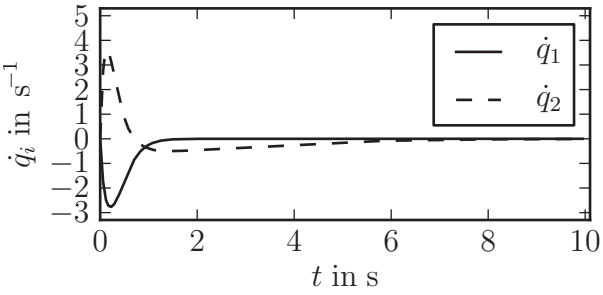


Fig. 5. Angular velocities of the underactuated manipulator.

6. SUMMARY AND OUTLOOK

We suggested a method for a direct simulation of holonomic mechanical systems based on its Lagrangian.

The derivatives required by the Euler-Langrange equations are calculated using algorithmic differentiation.

In principle, this approach can be extended to nonholonomic systems [20], because the same types of derivatives are required. However, the systems dynamics are formulated in terms of differential-algebraic equations, whose numerical solution is much more complicated [21, 22].

A similar concept was introducted in [23], where the mechanical system is formulated in terms of the Hamilton equations of motion. This approach requires only first order derivatives, but the usage of the Hamilton equations is less common in engineering.

7. REFERENCES

[1] W. Nolting, *Grundkurs Theoretische Physik 2, Analytische Mechanik*, Springer-Verlag, Berlin, 4th edition, 2004.

[2] D. D. Holm, T. Schmah, and C. Stoica, *Geometric Mechanicas and Symmetry: From Finite to Infinite Dimensions*, Oxford University Press, 2009.

[3] R. W. Brockett, "Control theory and analytical mechanics," in *Geometric Control Theory, Lie Groups: History, Frontiers, and Applications*, C. F. Martin and R. Hermann, Eds., vol. VII, pp. 1–46. Math Sci Press, Brookline, MA, 1976.

[4] J. Baillieul, "The geometry of controlled mechanical systems," in *Mathematical Control Theory*, J. Baillieul and J. C. Willems, Eds., chapter 9, pp. 322–354. Springer-Verlag, New York, 1998.

[5] A. J. van der Schaft, "System theory and mechanic," in *Three Decades of Mathematical System Theory*, H. Nijmeijer and J. M. Schumacher, Eds., vol. 135 of *Lecture Notes in Control and Information Science*. Springer, 1989.

[6] H. Nijmeijer and A. J. van der Schaft, *Nonlinear Dynamical Control systems*, Springer, New York, 1990.

[7] S. Wolfram, *The MATHEMATICA Book*, Cambridge University Press, 1999.

[8] J.-M. Cornil and P. Testud, *An Introduction to Maple V*, Springer, 2001.

[9] "Maxima, a computer algebra system," `http://maxima.sourceforge.net/`.

[10] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, 2nd edition, 2008.

[11] A. Walther, "Computing sparse hessians with automatic differentiation," *ACM Transactions on Mathematical Software (TOMS)*, vol. 34, no. 1, pp. 3:1–3:15, 2008.

[12] A. Griewank, D. Juedes, and J. Utke, "ADOL-C: A package for automatic differentiation of algorithms written in C/C++," *ACM Trans. Math. Software*, vol. 22, pp. 131–167, 1996.

[13] A. Walther, A. Griewank, and O. Vogel, "ADOL-C: Automatic differentiation using operator overloading in C++," *Proc. in Applied Mathematics and Mechanics*, vol. 2, no. 1, pp. 41–44, 2003.

[14] Ch. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland, "ADIFOR — Generating derivative codes from Fortran programs," *Scientific Programming*, vol. 1, no. 1, pp. 11–29, 1992.

[15] C. H. Bischof, H. M. Bücker, B. Lang, A. Rasch, and A. Vehreschild, "Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs," in *Proc. Second IEEE International Workshop on Source Code Analysis and Manipulation*. 2002, pp. 65–72, IEEE Computer Society.

[16] A. De Luca, S. Iannitti, R. Mattone, and G. Oriolo, "Underactuated manuipulators: Control properties and techniques," *Machine Intelligence and Robotic Control*, vol. 4, pp. 113–125, 2002.

[17] C. Knoll and K. Röbenack, "Sliding mode control of an underactuated two-link manipulator," *Proc. in Applied Mathematics and Mechanics*, vol. 10, no. 1, pp. 615–616, 2010.

[18] C. Knoll and K. Röbenack, "Control of an underactuated manipulator using similarities to the double integrator," in *Proc. 18th IFAC World Congress*, 2011.

[19] G. Oriolo and Y. Nakamura, "Control of mechanical systems with second-order nonholonomic constraints: Underactuated manipulators," in *Proc. of the 30th Conf. on Decision and Control*, Brighton, England, Dec. 1991, pp. 2398–2403.

[20] A. M. Block et al., *Nonholonomic Mechanics and Control*, Springer, 2003.

[21] E. Griepentrog and R. März, *Differential-Algebraic Equations and Their Numerical Treatment*, vol. 88 of *Teubner-Texte zur Mathematik*, Teubner Verlagsgesellschaft, Leipzig, 1986.

[22] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, SIAM, Philadelphia, 2nd edition, 1996.

[23] S. Palis and F. Palis, "Mechanical system simulation via automatic differentiation," in *Proc. of Advanced Problems of Mechanics*, 2010.