



Comparison of Automatic and Symbolic Differentiation in Mathematical Modeling and Computer Simulation of Rigid-Body Systems *

AXEL DÜRRBAUM, WILLY KLIER and HUBERT HAHN

Laboratory of Control and System Dynamics, Department of Mechanical Engineering,
University of Kassel, D-34109 Kassel, Germany

(Received: 23 November 1999; accepted in revised form: 29 January 2002)

Abstract. The objective of this paper is to check the efficiency and validity of two approaches for computing derivatives of complex functions, *automatic differentiation* using ADOL-C and *symbolic differentiation* using MACSYMA. This has been done in three benchmark examples, where the gradient of a Helmholtz energy function has been computed for different dimensions of independent variables (Example 1) and Jacobian matrices of inverse kinematics of planar and spatial parallel robots (Examples 2 and 3) have been computed. The results have been evaluated under six criteria: preliminary implementation work, computation time, flexibility in applications, limits of applicability, accuracy, and memory requirements.

ADOL-C was superior to MACSYMA concerning *preliminary work* (programming, source code generation, and compilation) and *modifications* of the functions to be differentiated and the differentiation task to be performed. In addition, contrary to MACSYMA, no *limits of applicability* were observed for ADOL-C, even in the simulation of complex multi-body systems.

On the other hand, for ADOL-C the *computation time* of derivatives was 10 to 40 times higher than for MACSYMA. As a consequence, differentiation by MACSYMA is better suited for real-time applications like hardware in the loop simulation, real-time control and real-time data processing than ADOL-C.

Both programs provide numerical results of equal *accuracy*.

Key words: automatic differentiation, kinematics of parallel robots, rigid-body systems, benchmark examples.

1. Introduction

In various tasks from the areas of computer simulation, control, measurement and experimental identification of spatial rigid-body systems and mechanisms, partial derivatives of lengthy and complex nonlinear symbolic expressions as, for example,

- Jacobian matrices of kinematic relations,

* This work has been supported by the German Science Foundation (DFG) under contract No. Ha 1666/6-1.

- Lie derivatives and Lie brackets used in feedback linearization algorithms, and
- Jacobian matrices occurring in parameter identification and sensitivity analysis

must be computed. The computational work to differentiate those rather complex expressions may be very cumbersome, time consuming, and sensitive with respect to numerical errors. It may even provide simulation models and control as well as measurement algorithms that can not be implemented in real-time.

Different approaches exist to perform and approximate derivatives of functions:

- *symbolic differentiation* using Computer Algebra Systems (CAS),
- *numerical differentiation* by means of divided differences,
- *processing of functions through high frequency filter algorithms*, etc.

In realistic theoretical models of spatial motions of rigid-body systems, extreme lengthy expressions may occur that cannot be computed even by efficient *computer algebra systems*. The accuracy of divided differences used in *numerical differentiation* may be hard to assess, and numerical errors tend to grow with problem complexity. Differentiation by means of *high frequency filtering* only provides *time derivatives* of functions.

A new approach to solve this task is called *Automatic Differentiation (AD)*. Its theoretical basis was developed in the late 70s [31, 35, 44]. Several program packages for AD have been developed, a.o. ADIFOR [1], ADIC [3] and ADOL-C [12, 13] (http://www.mcs.anl.gov/autodiff/adtools/AD_Tools/index.html contains a comprehensive list of AD-tools). These differentiation programs have been applied to various scientific and engineering tasks such as, e.g., nonlinear optimization [10, 26, 43], weather forecast [32, 42], astrodynamics [27], and control [5, 36].

The purpose of this paper is to compare the efficiency and applicability of two differentiation approaches

- ADOL-C for Automatic Differentiation (AD) and
- MACSYMA for symbolic differentiation (CAS)

by means of three benchmark examples. After a brief review of automatic differentiation in Section 2, the results will be judged by means of six evaluation criteria introduced in Section 3. The two differentiation methods MACSYMA and ADOL-C will be applied to the following examples in Section 4:

- computation of the gradient of a Helmholtz energy function for different numbers of independent variables (Example 1, Section 4.1) and

- computation of Jacobian matrices of inverse kinematics of a planar parallel robot (Example 2, Section 4.2) and of a spatial parallel robot (Example 3, Section 4.2)

The results obtained by computing derivatives of functions of these examples with ADOL-C and MACSYMA provide strong hints about where to apply *automatic differentiation* and where to apply *symbolic differentiation*.

2. Brief Review of Automatic Differentiation

Automatic differentiation is based on the fact that on a computer, every (complex) function $F : x \in \mathbb{R}^n \mapsto y = F(x) \in \mathbb{R}^m$, with x and y as vectors of *independent* and *dependent* variables, will be computed by elementary arithmetic operations (addition, multiplication, sin, etc.). With repeated application of the chain rule of derivative calculus to compositions of those elementary operations, numerical values of derivatives can be computed up to machine precision. The functions to be differentiated may include branches, loops, and subroutines, as well as intermediate variables.

For propagation of derivatives using chain rule, there exist two different techniques, called *forward* and *reverse* mode. Forward mode is efficient for a few independent variables and needs less memory, reverse mode is efficient for a few dependent variables but needs more memory. Various methods are available to implement automatic differentiation in a computer program [25]. Two such commonly used methods are *source code transformation* and *overloading of arithmetic operators*. In source code transformation, compiler techniques are used to transform the program source code into a new source code that computes the desired derivatives, whereas in operator overloading, the basic arithmetic operators of a programming language are overloaded and the desired derivatives are computed by tracing the computation of the expressions to be differentiated. ADOL-C [13] is a member of the latter class and has been chosen because it is capable of computing derivatives of an order higher than two (used in model-based control of mechatronic systems), only needs minor manual changes of the expressions to be differentiated, and requires less memory compared to members of source code transformation, like, e.g., ADIFOR. It performs the task of automatic differentiation by

- using C/C++ code and introducing a new data type for dependent and independent variables with *operator overloading*,
- allowing differentiation of arbitrary nested and recursive functions,
- calling supporting routines for calculating desired derivatives of any order,
- using univariate Taylor expansions, truncated after highest derivative degree d (specified by the user),

- recording the execution trace of the original evaluation program in a so-called *tape* (in memory and/or disk), and
- propagating the truncated Taylor series for each operation on the tape in *forward mode* and the corresponding adjoint operations on adjoint variables in *reverse mode*.

3. Evaluation Criteria of MACSYMA and ADOL-C

Computer Algebra Systems (CAS) are common tools for computing derivatives of functions in mathematical modeling of rigid-body systems [14, 38–41] and in nonlinear control of those systems [19, 20, 37]. Practical applications to complex industrial processes show limits of applicability of CAS (concerning source code generation and memory requirements). There are several benchmark tests that compare automatic differentiation with standard differentiation methods. In an investigation of [11], the authors come to the conclusion that ADOL-C is superior* to CAS MACSYMA with respect to implementation work and computation time. In comparison checks of ADIFOR with divided differences in [1, 2] and of TAMC with hand-coded derivatives in [9] the authors come to the conclusion that AD tools are at least equivalent to standard differentiation methods with respect to computation time and numerical accuracy. The results obtained in [1, 2, 9] differ from the results presented in [4] and also from the results obtained by the authors of this paper in recent computations of derivatives of functions used in computer simulations, experimental identification, and nonlinear control of parallel robots [19, 20, 22]. Some of these results obtained by applying a representative of automatic differentiation (ADOL-C) and a representative of CAS (MACSYMA) to three benchmark examples will be presented in Section 4. The results obtained will be judged by the following evaluation criteria:

I. Amount of preliminary implementation work

The amount of preliminary implementation work is defined as the time required to obtain executable programs for computing derivatives of functions. This preliminary implementation work includes three steps:

- Step 1: amount of time to program the function to be differentiated (t_{prog}),
- Step 2: amount of time to generate source code for computing derivatives (t_{source}),
and
- Step 3: amount of time to compile and link the source code (t_{comp}).

* Using a PC under MS-DOS with insufficient memory.

II. *Computation time*

Computation time is defined as the time required to provide numerical values of derivatives of functions to be differentiated (t_{run}). This criterion is important with respect to real-time implementation of computer simulation programs, of control algorithms and signal processing algorithms.

III. *Flexibility in the applications*

Flexibility in the applications is defined as measure of the amount of work, needed to obtain executable programs for computing derivatives when modifications of the functions to be differentiated or modifications of the differentiation task are required.

IV. *Limits of applicability*

Limits of applicability of differentiation methods may depend on the complexity of the functions to be differentiated, on the amount of virtual memory needed, and on the capability of the compilers to handle large numbers of instructions needed in the differentiation task.

V. *Accuracy of the results*

Computational accuracy is defined as difference between numerical values of derivatives obtained by applying MACSYMA and ADOL-C.

VI. *Memory requirements*

The memory requirements are measured both in terms of the amount of dynamic memory needed for storing variables and in terms of the size of source code needed to compute the desired derivatives (both measured in KB).

4. **Examples**

Three examples will be discussed in this section. The first one (Section 4.1) is a formal benchmark example. The next two examples in Section 4.2 are practical applications from the area of rigid-body systems (parallel robots).

ADOL-C functions `gradient` and `Jacobian` will be used in computations of derivatives by means of automatic differentiation.

Computation of derivatives by symbolic differentiation will be performed using a source code generated by MACSYMA. In the case of complex functions the MACSYMA command `diff` yields extreme large source code. By applying the MACSYMA command `optimize` on generated symbolic derivatives, all terms will be simplified, and *substitution variables* will be introduced which avoid re-computation of common subexpressions. This provides a much shorter source code with a shorter compilation time and a faster computation time. In this paper, the

Table I. Steps performed in the benchmark examples.

CAS: MACSYMA	AD: ADOL-C
M1: Programming of a MACSYMA command file including functions to be differentiated.	A1: Programming of functions to be differentiated as C++ module.
M2: Source code generation of derivative expressions.	A2: Inclusion of the ADOL-C subroutine into the main program.
M2.1: choice of dependent and independent variables.	A2.1: choice of dependent and independent variables.
M2.2: computation of nonsubstituted symbolic derivatives of functions.	A2.2: marking of functions to be differentiated.
M2.3: computation of substituted symbolic derivatives of functions.	A2.3: inclusion of C++ modules into main program.
M2.4: storage of generated source code in FORTRAN.	A2.4: choice of differentiation tasks.
M2.5: conversion of FORTRAN source code into C source code.	A3: Compilation of main program.
M3: Inclusion of source code of derivative expressions into the main program.	A3.1: choice of a computer system.
M4: Compilation of the main program.	A3.2: choice of compiler settings.
M4.1: choice of a computer system.	A4: Execution of the benchmark program.
M4.2: choice of compiler settings.	
M5: Execution of the benchmark program.	

output of the command `diff` will be called a *nonsubstituted* source code, and the output of the command `optimized` will be called a *substituted* source code.

The computation time of executable programs depends, a.o., on the compiler settings used. In all three benchmark examples, only standard compiler optimizations were used, although this may sometimes lead to compilation problems (cf. Table III, remarks 1, 2). The programs for computing derivatives of Examples 1, 2, and 3 were generated and executed by performing the steps of Table I.

4.1. BENCHMARK EXAMPLE 1 (HELMHOLTZ ENERGY FUNCTION)

In a first step, the two methods, symbolic differentiation with MACSYMA and automatic differentiation with ADOL-C, were applied to the following example [11]. The gradient of a function (Helmholtz energy function)

$$f(x) = R T \sum_{i=1}^n x_i \log \frac{x_i}{1 - b^T x} - \frac{x^T A x}{\sqrt{8} b^T x} \log \frac{1 + (1 + \sqrt{2}) b^T x}{1 + (1 - \sqrt{2}) b^T x} \quad (1)$$

with

$$0 \leq b, \quad 0 \leq x, \quad b, x = (x_1, \dots, x_n)^T \in \mathbb{R}^n, \quad A = A^T \in \mathbb{R}^{n \times n}$$

has been computed for different numbers of independent variables x_i ($n = 1, 2, 5, 10, 20, 50$).

This benchmark example has been computed according to the steps of Table I with the following details: Step M2: the source code of symbolic derivatives has been generated on a LINUX system (compare Table II). Steps M2.2, M2.3: both, nonsubstituted and substituted symbolic expressions were used (compare Table II). Steps M4, A3: the benchmark program has been completely written in C++. Steps M4.1, A3.1: it has been executed on three typical computer systems (compare Table IV). Step M4.2, A3.2: the standard optimizations settings of each compiler have been used to compile and link the benchmark program (compare Table IV). Steps M5, A4: due to the fact that, in a simulation program, a large number of derivatives is computed in each simulation run (e.g., in Example 3, more than one million), the time needed for computing 10^6 derivatives has been measured (compare Table V and Figure 1).

The results obtained in this example will be now evaluated according to the evaluation criteria of Section 3.

I. Amount of preliminary implementation work

To Step 1: (programming time)

ADOL-C: Programming of the function $f(x)$ took some minutes ($t_{\text{prog}} \leq 5$ min).

MACSYMA: Symbolic differentiation has been done with MACSYMA 5.2 on a LINUX system (Table IV). Programming of the command file took some minutes ($t_{\text{prog}} \leq 5$ min).

To Step 2: (time for source code generation)

ADOL-C: No source code of derivative expressions had to be generated ($t_{\text{source}} = 0$ min).

MACSYMA: The amount of time for source code generation was $t_{\text{source}} = t_{\text{diff}} + t_{\text{norm}}$ in the nonsubstituted case, and $t_{\text{source}} = t_{\text{diff}} + t_{\text{sub}}$ in the substituted case, where t_{diff} , t_{norm} and t_{sub} are collected for different dimensions n of x in Table II. For large n ($n = 50$), this took the LINUX system $t_{\text{source}} = 40.3$ min in the nonsubstituted case, and $t_{\text{source}} = 120.8$ min in the substituted case.

To Step 3: (compilation time)

ADOL-C: The compilation time of the program ($1.01 \text{ s} \leq t_{\text{comp}} \leq 3.21 \text{ s}$) was negligible compared to the compilation time of MACSYMA source code and independent of the dimension n (compare Table III, lines 'adlc').

Table II. Source code generation using MACSYMA.

n	1	2	5	10	20	50
t_{diff} [sec]	0.1	0.2	0.22	1.07	7.02	175.65
t_{non} [sec]	0.73	0.24	2.04	7.34	52.20	2241.97
size [byte]	740	2385	15561	83414	558544	7692844
lines [-]	11	31	196	1041	6871	92951
t_{sub} [sec]	0.64	0.46	3.01	16.22	134.50	7070.31
size [byte]	650	1082	3351	10622	39372	242769
lines [-]	20	27	66	200	710	4318
n_s [-]	15	19	37	107	397	2467

n	number of independent variables x_i
t_{diff}	time to compute differentiations $\partial f(x)/\partial x_i$
t_{non}	time to create nonsubstituted MACSYMA source code
t_{sub}	time to create substituted MACSYMA source code
size	size of source code in bytes
lines	number of source code lines
n_s	number of used substitution variables

MACSYMA: The compilation time of this example has grown tremendously with increasing dimension n (compare Table III). It was much higher in the nonsubstituted case for the SGI system ($t_{\text{comp}} = 8.7$ min) than in the substituted case ($t_{\text{comp}} = 50$ sec for as, e.g., $n = 20$).

In summary, the results with respect to criterion 1 were that the amount of work to program the function $f(x)$ to be differentiated was identical for ADOL-C and MACSYMA; the amount of time for source code generation of derivatives was much higher for MACSYMA than for ADOL-C; and the compilation time for MACSYMA was much higher than the compilation time for ADOL-C. For MACSYMA it grew with increasing n (ratios of 1000:1 have been observed for different n). For ADOL-C it was independent of n .

II. Computation time

ADOL-C: Computation of derivatives using ADOL-C was much slower than using MACSYMA (compare Figure 1).

MACSYMA: The computation time of source code generated by MACSYMA was 10 to 66 times faster compared to ADOL-C.

Table III. Compilation time in seconds.

system	method	$n =$	1	2	5	10	20	50
LINUX	non		0.96	1.28	4.54	33.68	1	1
	sub		0.97	1.02	1.47	4.01	25.25	628.34
	adolc		1.11	1.08	1.05	1.01	1.03	1.02
IBM	non		1.34	1.19	2.32	14.97	490.95	1
	sub		1.28	1.34	1.46	2.56	24.93	833.65
	adolc		1.34	1.41	1.42	1.47	1.33	1.42
SGI	non		2.60	3.14	7.63	42.93	522.16	2
	sub		2.68	2.89	4.22	11.54	50.01	94.36
	adolc		3.17	3.17	3.18	3.21	3.20	3.19

system computer system used (compare Table IV)

method differentiation method

non symbolic differentiation using nonsubstituted MACSYMA source

sub symbolic differentiation using substituted MACSYMA source

adolc automatic differentiation using ADOL-C

1 compiler error: *out of virtual memory*

2 compiler error: *too much operations per statement*

Table IV. Computer systems used.

system	LINUX	IBM RS6000/590	SGI SC900/4
CPU	AMD K6-2 300MHZ	8 PowerPC J50	4 MIPS 8000 SSR
RAM	96 MB	2 GB	1 GB
OS	Linux Debian 2.1	AIX 2.4	IRIX 6.5
compiler	egcs-2.91.60	x1C 3.1	CC 7.2.1.3m
optimization	-O2	-O2	O2 -IPA -LNO

III. Flexibility in the applications

ADOL-C: For each variation of $f(x)$ only parts of the preliminary implementation work must be repeated: reprogramming of $f(x)$ and recompilation of this program.

MACSYMA: For each variation of $f(x)$ the whole preliminary implementation work must be repeated: command file reprogramming, source code generation of derivatives and compilation of the program, where source code generation turned out to be very time consuming.

Table V. Time in seconds for computing 10^6 derivatives.

system	method	$n =$	1	2	5	10	20	50
LINUX	non		9.1	9.0	10.8	18.2	1	1
	sub		6.7	4.0	2.8	2.9	4.0	16.2
	adolc		224.2	124.9	80.3	84.4	123.3	268.8
IBM	non		10.9	10.9	10.8	17.2	55.2	1
	sub		6.8	3.5	2.2	1.8	2.8	21.92
	adolc		195.4	124.7	101.1	123.0	187.1	450.5
SGI	non		36.1	28.0	23.6	29.2	59.6	2
	sub		29.1	15.9	7.5	6.1	7.1	21.5
	adolc		447.4	274.8	214.2	262.5	418.8	925.9

system computer system used (compare Table IV)
 method differentiation method
 non symbolic differentiation using nonsubstituted MACSYMA source
 sub symbolic differentiation using substituted MACSYMA source
 adolc automatic differentiation using ADOL-C
 1 compiler error: *out of virtual memory*
 2 compiler error: *too much operations per statement*

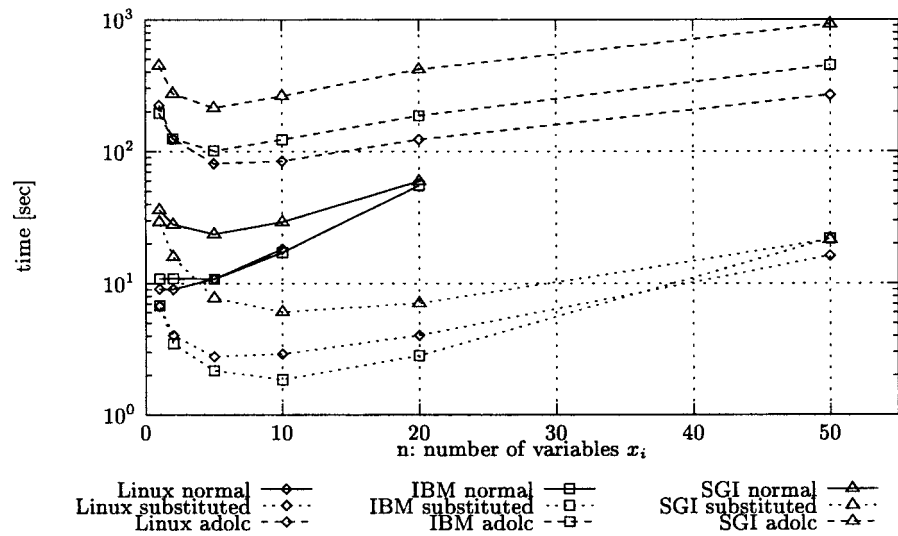


Figure 1. Time for computing 10^6 derivatives.

As a consequence, program modifications were much less expensive for ADOL-C than for MACSYMA (for instance, in the case of $n = 20$ about 3 min for MACSYMA and only 7 sec for ADOL-C).

IV. Limits of applicability

ADOL-C: In this example no limits of applicability have been observed even for large dimensions ($n = 200$).

MACSYMA: Large dimensions ($n \geq 50$) could not be handled due to compiler limitations or shortage of virtual and real memory (compare Table V, remarks 1, 2).

This implies that the limits of applicability have not been observed by applying ADOL-C. However, computations were limited to $n \leq 50$ for MACSYMA.

V. Accuracy of the results

In this benchmark example, the numerical results obtained by MACSYMA and ADOL-C were identical.

VI. Memory requirements

ADOL-C: Only the Helmholtz function to be differentiated had to be programmed. This took 25 lines of C source code and requires $n^2 + 3 * n$ double variables (about 20 KB of dynamic memory).

MACSYMA: When using nonsubstituted symbolic expressions to compute the gradient for $n = 50$ variables, up to 92951 lines of source code were required. The dynamic memory requirements were the same as for ADOL-C.

In the substituted case only 4318 lines of source code were required, but the dynamic memory requirements grew up to $n^2 + 3 * n + n_s$ double variables (about 41 KB), due to the 2467 substitution variables needed (compare Table II).

Computing derivatives using ADOL-C require less source code and memory with respect to MACSYMA. Using substituted symbolic expressions reduces the source code size compared to the nonsubstituted expressions at the cost of a slightly higher amount of dynamic memory.

The results obtained in Example 1 are summarized in Table VI.

4.2. EXAMPLES 2 AND 3 OF RIGID-BODY SYSTEMS

In this section, automatic differentiation is applied to two examples of rigid-body systems from industrial practice, including a planar parallel robot (Example 2, Figure 2), and a spatial parallel robot (Example 3, Figure 3). In computer simulations and in transformations of measured variables of those systems, derivatives of kinematic relations will be computed using two approaches:

Table VI. Results of Example 1.

Evaluation criterion	MACSYMA		ADOL-C
	nonsubstituted	substituted	
I. amount of preliminary work			
I.1 programming time		nearly the same	
I.2 source code generation time	long	long	–
I.3 compilation time	long	long	short
II. computation time	short	(1:10 to 66)	long
III. flexibility in application		not flexible	flexible
IV. limits of applicability		limited to dimensions $n \leq 50$	no limitations observed for $n \leq 200$
V. accuracy of results		identical numerical results	
VI. amount of memory (code lines)	large (92951)	small (4318)	very small (25)

- *Symbolic differentiation* by means of the computer algebra system MACSYMA and
- *Automatic differentiation* by means of an ADOL-C library.

The results of these computations will be compared according to the evaluation criteria of Section 3.

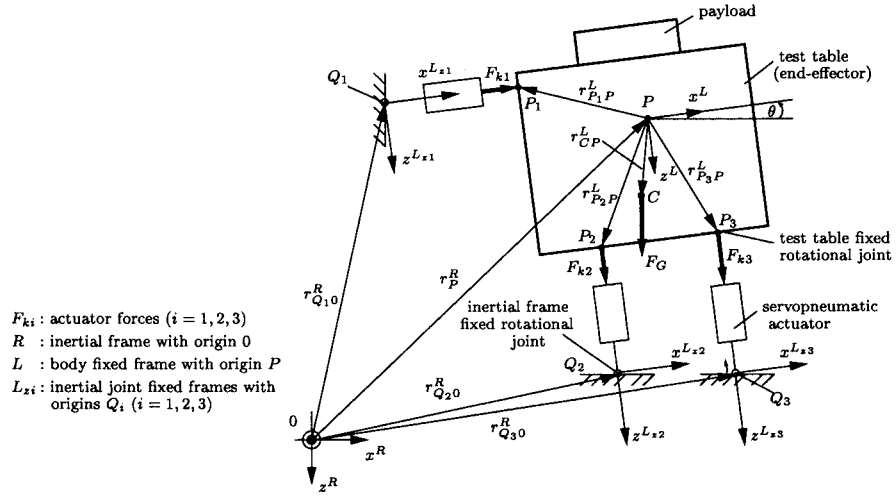
4.2.1. Kinematics of Planar and Spatial Parallel Robots

Kinematics of parallel robots will be considered from two aspects:

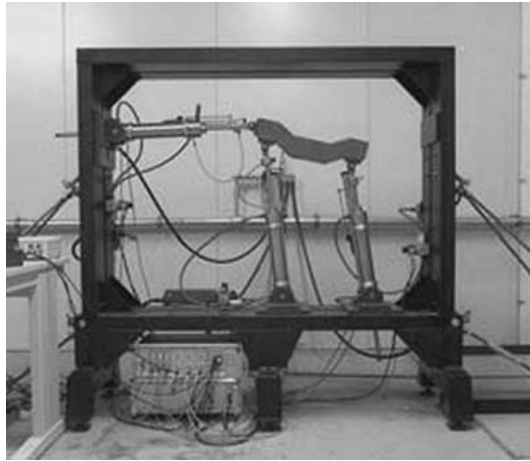
- *Direct kinematics*: finding absolute coordinates of the end-effector for given joint coordinates, and
- *The inverse kinematics*: finding joint coordinates for given absolute coordinates of the end-effector.

The solution of *inverse kinematics* of *parallel robots* is usually straightforward and can be obtained analytically [8, 20, 33, 45]. On the other hand, the solution of *direct kinematics* of *parallel robots* can only be obtained numerically [24, 30, 34].

Automatic Differentiation



(a) Vector diagram



(b) Photograph

Figure 2. Vector diagram and photograph of the planar multi-axis test facility built in the Laboratory of Control and System Dynamics.

The *parallel robots* considered in this paper are multi-axis test facilities for vibration tests. They include a rigid test table (rigid end-effector) attached to the base by three (planar case) or by six (spatial case) servopneumatic actuators with position and orientation vector p_e of the end-effector

$$\begin{aligned}
 p_e &:= [x_{P0}^R, z_{P0}^R, \theta]^T \in \mathbb{R}^3 \quad (\text{planar case}), \\
 p_e &:= [x_{P0}^R, y_{P0}^R, z_{P0}^R, \varphi, \theta, \psi]^T \in \mathbb{R}^6 \quad (\text{spatial case}),
 \end{aligned} \tag{2}$$

and distance z_{k_i} of base point Q_i to an attachment point P_i , represented in actuator housing fixed frames L_{z_i} (cf. Figures 2 and 3). In this section, the *Jacobian matrices of inverse kinematics* of the above parallel robots will be computed using *symbolic and automatic differentiation*. These Jacobian matrices are included

- in computations of *transformation matrices* that map actuator forces F_k , represented in joint space, into forces F_e and torques M_e , represented in task space (end-effector absolute coordinates), used in *computer simulations* (cf. Figure 4a), and
- in iterative numerical computations of *direct kinematics*, used in *transformations of measurements signals* [23] and in *control algorithms* [19, 20], (cf. Figure 4b).

Inverse kinematic relations of multi-axis test facilities (cf. Figures 2 and 3) have been modeled by the following expressions [17, 21, 28, 29]:

$$z_k = \begin{bmatrix} z_{k_1} \\ \vdots \\ z_{k_n} \end{bmatrix} = t_t(p_e)$$

$$:= \begin{bmatrix} t_{t_1}(p_e) \\ \vdots \\ t_{t_n}(p_e) \end{bmatrix} \in \mathbb{R}^n \quad \text{with} \quad n = \begin{cases} 3 & \text{in planar case,} \\ 6 & \text{in spatial case,} \end{cases} \quad (3)$$

with

$$t_{t_i} := -\left((x_{P_0}^R + \cos \theta \cdot x_{P_i P}^L + \sin \theta \cdot z_{P_i P}^L - x_{Q_{i0}}^R)^2 + (z_{P_0}^R - \sin \theta \cdot x_{P_i P}^L + \cos \theta \cdot z_{P_i P}^L - z_{Q_{i0}}^R)^2\right)^{1/2} \quad (4)$$

(planar case, $i = 1, 2, 3$) and

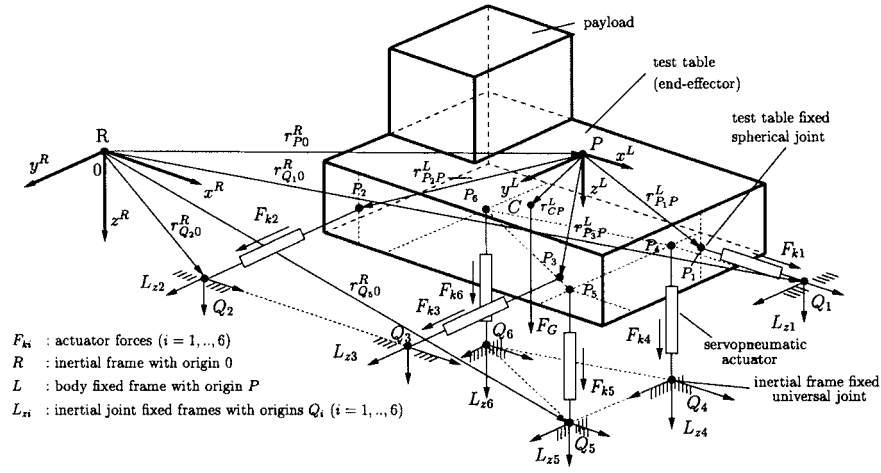
$$t_{t_i} := -\left((x_{P_0}^R + \cos \theta \cdot \cos \psi \cdot x_{P_i P}^L - \cos \theta \cdot \sin \psi \cdot y_{P_i P}^L + \sin \theta \cdot z_{P_i P}^L - x_{Q_{i0}}^R)^2 + (y_{P_0}^R + (\cos \varphi \cdot \sin \psi + \sin \varphi \cdot \sin \theta \cdot \cos \psi) \cdot x_{P_i P}^L - y_{Q_{i0}}^R - \sin \varphi \cdot \cos \theta \cdot z_{P_i P}^L + (\cos \varphi \cdot \cos \psi - \sin \varphi \cdot \sin \theta \cdot \sin \psi) \cdot y_{P_i P}^L)^2 + (z_{P_0}^R + (\sin \varphi \cdot \sin \psi - \cos \varphi \cdot \sin \theta \cdot \cos \psi) \cdot x_{P_i P}^L + \cos \varphi \cdot \cos \theta \cdot z_{P_i P}^L - z_{Q_{i0}}^R + (\sin \varphi \cdot \cos \psi + \cos \varphi \cdot \sin \theta \cdot \sin \psi) \cdot y_{P_i P}^L)^2\right)^{1/2} \quad (5)$$

(spatial case, $i = 1, \dots, 6$).

The inverse kinematic relations (4) and (5) include components of the vectors

$$r_{P_i P}^L := [x_{P_i P}^L, 0, z_{P_i P}^L]^T \in \mathbb{R}^3 \quad (\text{planar case}),$$

$$r_{P_i P}^L := [x_{P_i P}^L, y_{P_i P}^L, z_{P_i P}^L]^T \in \mathbb{R}^3 \quad (\text{spatial case}), \quad (6)$$



(a) Vector diagram



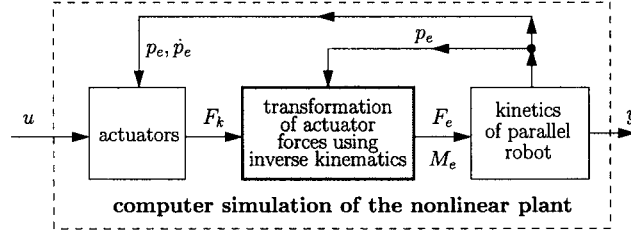
(b) Photograph

Figure 3. Vector diagram and photograph of the spatial multi-axis test facility built in the Laboratory of Control and System Dynamics.

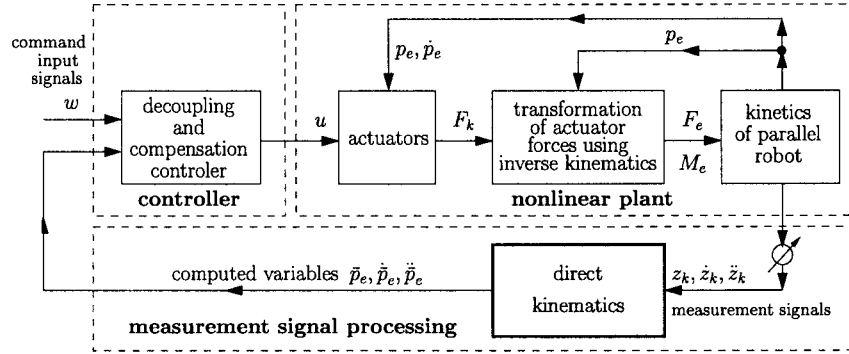
from reference point P to an attachment point P_i , represented in body fixed frame L , and components of the vectors

$$\begin{aligned}
 r^R_{Q_i,0} &:= [x^R_{Q_i,0}, 0, z^R_{Q_i,0}]^T \in \mathbb{R}^3 \quad (\text{planar case}), \\
 r^R_{Q_i,0} &:= [x^R_{Q_i,0}, y^R_{Q_i,0}, z^R_{Q_i,0}]^T \in \mathbb{R}^3 \quad (\text{spatial case}),
 \end{aligned}
 \tag{7}$$

from origin 0 of frame R to a base point Q_i , represented in inertial frame R . The transformation matrices of actuator forces F_k from joint space into task space (R, L)



(a) Block diagram of a computer simulation model of a robot including inverse kinematic relations



(b) Parallel robot control using a decoupling and compensation controller with nonlinear measurement variable processing based on direct kinematics

Figure 4. Block diagrams of mathematical models of systems that include inverse and direct kinematic relations.

$$\begin{bmatrix} F_e^R \\ M_e^L \end{bmatrix} = \begin{cases} J_p^T(p_e) \cdot F_k \in \mathbb{R}^3 & \text{(planar case),} \\ [J_p(p_e) \cdot T_e(p_e)]^T \cdot F_k \in \mathbb{R}^6 & \text{(spatial case),} \end{cases} \quad (8)$$

include the *Jacobian matrix of inverse kinematics*

$$J_p(p_e) := \frac{\partial t_t(p_e)}{\partial p_e} \in \mathbb{R}^{n,n} \quad \text{with } n = \begin{cases} 3 & \text{in planar case,} \\ 6 & \text{in spatial case.} \end{cases} \quad (9)$$

In the spatial case $T_e(p_e)$ is the matrix of the kinematic differential equation

$$\dot{p}_e := \begin{bmatrix} I_3 & & 0_{3,3} \\ 0_{3,3} & \underbrace{\begin{bmatrix} \frac{\cos \psi}{\cos \theta} & \frac{-\sin \psi}{\cos \theta} & 0 \\ \sin \psi & \cos \psi & 0 \\ -\cos \psi \cdot \frac{\sin \theta}{\cos \theta} & \sin \psi \cdot \frac{\sin \theta}{\cos \theta} & 1 \end{bmatrix}}_{:=T_e(p_e)} & \end{bmatrix} \cdot v_e \quad (10)$$

with the end-effector velocity vector

$$v_e := [\dot{x}_{P0}^R, \dot{y}_{P0}^R, \dot{z}_{P0}^R, \omega_x^L, \omega_y^L, \omega_z^L]^T \in \mathbb{R}^6 \quad \text{(spatial case).} \quad (11)$$

In *direct kinematics*, absolute coordinates p_e of the end-effector are computed from measured joint coordinates z_k . This is much easier than measurement of absolute coordinates p_e of the end-effector. The control space of multi-axis test facilities is usually represented in absolute coordinates (controlled variables p_e) of the end-effector (cf. Figure 4b). As a consequence, in most control and measurement tasks, end-effector variables p_e must be computed from measured variables z_{ki} .

In the investigation considered, the absolute coordinate vector p_e has been obtained from relation (3) according to the *direct kinematic equation* (Newton–Kantorowitch iteration)

$$p_e(i) = p_e(i-1) + \alpha_d \cdot [J_p(p_e)]_{p_e(i-1)}^{-1} \cdot (t_i(p_e(i-1)) - z_k),$$

(iteration step i , numerical damping coefficient α_d). (12)

It includes the *Jacobian matrix of inverse kinematics* $J_p(p_e)$.

4.2.2. Results

In this section, the results obtained by symbolic and automatic differentiation of inverse kinematic relations of planar and spatial parallel robots will be compared. The systems considered are multi-axis test facilities, driven by earthquake and sine sweep (1–20 Hz) command input signals, and controlled by feedback linearization algorithms [15, 16, 20].

The benchmark programs of Examples 2 and 3 have been performed following the steps of Table I. The controlled multi-axis test facilities have been simulated on a LINUX system using a C computer simulation program (Steps M4.1 and A3.1). The functions of these examples have been differentiated by MACSYMA with substituted source code (Step M2.3) and with standard compiler optimization (Step M4.2) and by ADOL-C with standard compiler optimization (Step A3.2), because the results of Example 1 show that these two settings provide rapid computation of derivatives.

The Jacobian matrices $J_p(p_e)$ have been computed up to three times (three iterations of (12)) in each integration step Δt ($\Delta t = 1/3000$ sec) for a simulated physical process of a period of 6 sec. This leads to about 54000 ($54000 = 3000 \times 6 \times 3$) computations of the Jacobian matrix for each simulation run. As a consequence, in each simulation run 486000 (54000×9) partial derivatives have been computed for the planar robot and 1944000 (54000×36) partial derivatives for the spatial robot (Steps M5 and A4).

The results obtained for Examples 2 and 3 will be now evaluated according to the evaluation criteria of Section 3.

I. Amount of preliminary implementation work

The first evaluation criterion discussed is related to preliminary implementation work, defined as the time required to obtain executable programs for calculating

the Jacobian matrices $J_p(p_e)$ of inverse kinematics of planar and spatial parallel robots.

To Step 1: (programming time)

ADOL-C: Automatic differentiation tool ADOL-C requires programming of functions to be differentiated ($z_k = t_t(p_e)$), choice of dependent variables (z_k) and independent variables (p_e), marking of functions to be differentiated, and splitting of subroutines into C++ modules (ADOL-C) and C modules (simulation program). These tasks, specially splitting of subroutines into C and C++ modules, took in both examples (planar and spatial robot) $t_{\text{prog}} \approx 30$ min.

MACSYMA: Preliminary work when applying MACSYMA includes programming of a command file with functions to be differentiated ($z_k = t_t(p_e)$) as well as the choice of dependent variables (z_k) and independent variables (p_e). These programming tasks, especially programming of the command file, took $t_{\text{prog}} \approx 35$ min in the planar case and $t_{\text{prog}} \approx 50$ min in the spatial case.

To Step 2: (time of source code generation)

ADOL-C: No source code of derivative expressions had to be generated ($t_{\text{source}} = 0$ min).

MACSYMA: Source code generation has been performed by computing symbolic derivatives of functions ($J_p(p_e)$), by computing substituted symbolic derivatives of functions, and by including symbolic derivatives into the simulation program. In both examples these tasks took a few minutes ($t_{\text{source}} \leq 7$ min).

To Step 3: (compilation time)

ADOL-C: The compilation time of the simulation program including ADOL-C subroutines was $t_{\text{comp}} \approx 92.2$ sec in the planar case and $t_{\text{comp}} \approx 138.1$ sec in the spatial case.

MACSYMA: The compilation time of the simulation program including symbolic Jacobian matrices $J_p(p_e)$ generated by MACSYMA was $t_{\text{comp}} \approx 93.0$ sec in the planar case and $t_{\text{comp}} \approx 138.4$ sec in the spatial case.

Summarizing, Examples 2 and 3 show that the implementation work is up to two times higher for MACSYMA than for ADOL-C. In agreement with Example 1, the programming time for ADOL-C is independent of the complexity of functions to be differentiated. In Examples 2 and 3, the programming time for MACSYMA increases with growing complexity of the functions to be differentiated. The time needed for source code generation is much higher for MACSYMA than for ADOL-C. The compilation times for Examples 2 and 3 are nearly identical for both differentiation methods. This does not agree with the results of Example 1, where a remarkable difference of the compilation times for ADOL-C subroutine and MACSYMA source code has been observed. This is due to the fact that in Examples 2 and 3 the functions to be differentiated were only a small part of the total simulation program to be compiled.

II. *Computation time*

ADOL-C: The computation time of the simulation program was $t_{\text{run}} \approx 2.45$ sec in the planar case, and $t_{\text{run}} \approx 16.26$ sec in the spatial case.

MACSYMA: The computation time of the simulation program including symbolic Jacobian matrices $J_p(p_e)$ was $t_{\text{run}} \approx 0.06$ sec in the planar case, and $t_{\text{run}} \approx 1.92$ sec in the spatial case. These results show in agreement with Example 1, that symbolic computation of Jacobian matrices $J_p(p_e)$ by MACSYMA is, in the planar case, about 40 times faster than by ADOL-C and, in the spatial case, about 8 times faster than by ADOL-C.

With increasing complexity of the functions to be differentiated, the number of operations of ADOL-C increases, too, whereas the number of statements in the source code generated by MACSYMA increases more rapidly. This implies that the major advantage of MACSYMA (brief computation time) with respect to ADOL-C slowly decreases with increasing complexity of the functions to be differentiated.

III. *Flexibility in the applications*

ADOL-C: In the case where the differentiation task changes and the model equations must be modified, only reprogramming of the inverse kinematic relations and recompilation of the simulation program must be performed.

MACSYMA: For each variation of the differentiation task and for each variation of the model equations, the whole preliminary implementation work must be repeated: reprogramming of the command file including inverse kinematic relations, generation of source code for computing Jacobian matrices, and compilation of the simulation program.

This implies in agreement with Example 1, that program modifications are less expensive for ADOL-C than for MACSYMA. A variation of the differentiation task or of the model equations in Examples 2 and 3 leads to an amount of work which is more than 2 times higher for MACSYMA than for ADOL-C.

IV. *Limits of applicability*

ADOL-C and MACSYMA: The complexity of the functions to be differentiated in Examples 2 and 3 does not demand too large memory of the computer system. The system used (LINUX) is in both examples (planar and spatial robot) for MACSYMA and ADOL-C capable of computing the desired derivatives.

In case of computer simulations of hyper-redundant actuator configurations [6] (cf. Figure 5a) as well as in computer simulations [7] and feedback linearization control [18] of spatial rigid-body systems including a large number of rigid bodies (cf. Figure 5b), derivatives required in simulation programs and in control algorithms could not be successfully computed by using MACSYMA. These tasks exceeded the limits of applicability of MACSYMA. On the other hand, the al-

Table VII. Memory requirements.

	MACSYMA	ADOL-C
memory required at runtime in bytes		
planar case	320	336
spatial case	928	944
size of the source code in KB		
planar case	1679	1175
spatial case	7618	4797

gorithms obtained by ADOL-C were slow. They could neither be used in hardware in the loop simulations nor in real-time control of those systems.

V. Accuracy of the results

The numerical values of elements of the Jacobian matrices computed by ADOL-C were nearly identical with the numerical values obtained by MACSYMA. The maximum numerical differences between these values were less than 10^{-16} .

VI. Memory requirements

The memory required at runtime and the size of the source code, measured in KB of the simulation program including ADOL-C subroutines and of the simulation program including symbolic Jacobian matrices $J_p(p_e)$ generated by MACSYMA are collected in Table VII.

Examples 2 and 3 show that the memory required at runtime is slightly higher for ADOL-C than for MACSYMA and that the size of the source code including the computation of partial derivatives is about 1.5 times larger for MACSYMA than for ADOL-C.

The results of Examples 2 and 3 concerning the evaluation criteria I to VI are summarized in Table VIII.

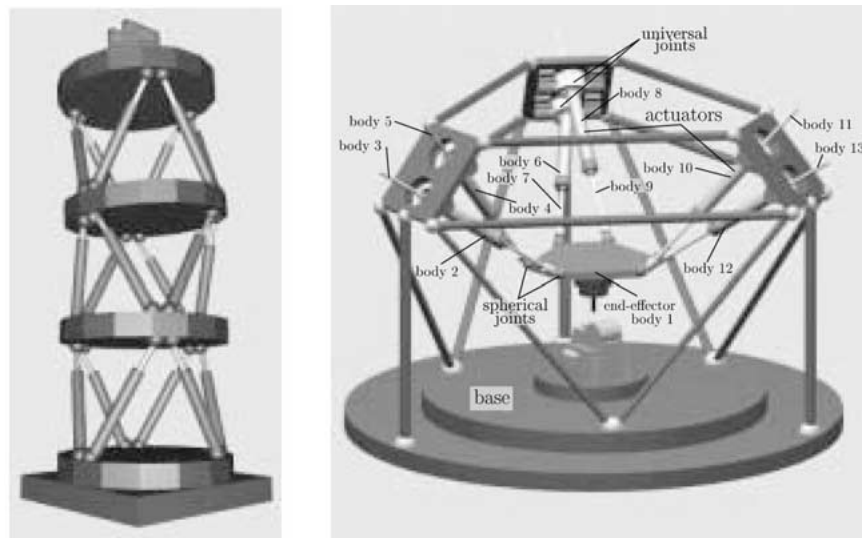
5. Conclusions

The results obtained in these investigations show that

- *Automatic differentiation* with ADOL-C is superior to symbolic differentiation by MACSYMA
 - in applications where extreme complex expressions must be differentiated (no limits of applicability have been observed for ADOL-C),

Table VIII. Results obtained for Examples 2 and 3.

Evaluation criterion	MACSYMA	ADOL-C
I. amount of preliminary work		
I.1 programming time		
– planar case		short (nearly the same)
– spatial case	long	short (same as planar)
I.2 source code generation time		
– planar case	long	–
– spatial case	long	–
I.3 compilation time		
– planar case		short (nearly the same)
– spatial case		long (nearly the same)
II. computation time		
– planar case	short (1:40)	long
– spatial case	short (1:8)	long
III. flexibility in application		
– planar case		
– spatial case	not flexible	flexible
IV. limits of applicability		
– planar case		
– spatial case		no limitations observed
V. accuracy of results		
– planar case		
– spatial case		identical numerical results
VI. memory requirements		
– memory required at runtime		
– planar case		very small (nearly the same)
– spatial case		very small (nearly the same)
– size of the source code		
– planar case	small (1.4:1)	small
– spatial case	medium (1.6:1)	medium



(a) hyper-redundant manipulator (b) spatial hexapod

Figure 5. Computer graphics of a hyper-redundant manipulator (a) and of a spatial hexapod including 13 rigid bodies (b).

- in tasks where preliminary work and amount of time to modify expressions to be differentiated, to generate and compile source code, and to modify the differentiation task, are of primary importance, as well as
 - in applications where no real time demands must be satisfied (for instance in the first design phase of rather complex mechanisms by means of computer simulations).
- Execution of differentiation is up to 40 times faster for *symbolic differentiation* than for *automatic differentiation*. This implies that *real-time applications* like
- hardware in the loop simulation,
 - real-time control, and
 - real-time data processing

symbolic differentiation is superior to automatic differentiation (if the expressions to be differentiated are not too complex to be handled by symbolic differentiation).

In current investigations of the authors, automatic differentiation is used for computing feedback linearization controllers where Lie-derivatives of complex functions of a large number of variables are required. This task could not yet be successfully solved by MACSYMA.

References

1. Bischof, C., Carle, A., Corliss, G., Griewank, A. and Hovland, P., 'ADIFOR – Generating derivative codes from Fortran programs', *Scientific Programming* **1**(1), 1992, 11–29.
2. Bischof, C., Carle, A., Khademi, P. and Mauer, A., 'ADIFOR 2.0: Automatic differentiation of Fortran 77 programs', *IEEE Computational Science and Engineering* **3**(3), 1996, 18–32.
3. Bischof, C. and Roh, L. and Mauer, A., 'ADIC – An extensible automatic differentiation tool for ANSI-C', *Software-Practice and Experience* **27**(12), 1997, 1427–1456.
4. Campbell, S.L. and Hollenbeck, R., 'Automatic differentiation and implicit differential equation', in *Computational Differentiation*, M. Berz, et al. (eds), SIAM, Philadelphia, PA, 1996, 215–227.
5. Campbell, S.L., Moore, E. and Zhong, Y., 'Utilization of automatic differentiation in control algorithms', *IEEE Transactions on Automatic Control* **39**, 1994, 1047–1052.
6. Chirikjian, G.S. and Burdick, J.W., 'A hyper-redundant manipulator', *IEEE Robotics and Automation Magazine* **1**(4) 1995, 22–29.
7. Fürst, D., Hecker, F. and Hahn, H., 'Mathematical modeling and parameter identification of a planar servo-pneumatic test facility, Part I: Mathematical modeling and computer simulation', *Nonlinear Dynamics* **14**, 1997, 249–268.
8. Geng, Z., Hayes L., Lee, J.D. and Carroll, R., 'On the dynamic model and kinematic analysis of a class of Stewart platforms', *Robotics and Autonomous Systems* **9**, 1992, 237–254.
9. Giering, R. and Kaminski, T., 'Comparison of automatically generated code for evaluation of first and second order derivatives to hand written code from the Minpack-2 collection', in *Automatic Differentiation for Adjoint Code Generation*, C. Faure (ed.), INRIA, Nancy, France, 1998, 31–38.
10. Grandinetti, L. and Conforti, D., 'Numerical comparisons of nonlinear programming algorithms on serial and vector processors using automatic differentiation', *Mathematical Programming* **42**, 1988, 375–389.
11. Griewank, A., 'On automatic differentiation', in *Mathematical Programming: Recent Developments and Applications*, M. Iri and K. Tanabe (eds), Kluwer Academic Publishers, Dordrecht, 1989, 83–108.
12. Griewank, A., *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Frontiers in Applied Mathematics, Vol. 19, SIAM, Philadelphia, PA, 2000.
13. Griewank, A., Juedes, D., Srinivasan, J. and Tyner, C., 'ADOL-C, A package for the automatic differentiation of algorithms written in C/C++', *ACM Transactions on Mathematical Software* **22**(2), 1996, 131–167 (url: <http://www.math.uni-dresden.de/adolc>).
14. Hahn, H. and Fürst, D. and Hecker, F., 'Theoretical modeling of multi-axis test facilities – A new approach', in *Abstracts of the 19th Conference of Theoretical and Applied Mechanics*, Kyoto, Japan, T. Tatsumi, E. Watanabe and T. Kambe (eds), Elsevier Science Publishers, Amsterdam, 1996, 675.
15. Hahn, H. and Klier, W., 'Nonlinear control of multi-axis test facilities with redundant servopneumatic actuators', RTS-Bericht RT-25, Fachgebiet Regelungstechnik und Systemdynamic (Maschinenbau), Universität-Gh Kassel, 1998.
16. Hahn, H. and Klier, W., 'Nonlinear control of planar robots with redundant servopneumatic actuators', RTS-Bericht RT-24, Fachgebiet Regelungstechnik und Systemdynamic (Maschinenbau), Universität-Gh Kassel, 1998.
17. Hahn, H. and Klier, W., 'Nonlinear control of spatial parallel robots with redundant servopneumatic actuators', in *Proceedings of the International Conference on Systems, Signals, Control, Computers*, Durban, South Africa, Vol. II, V.B. Bajic (ed.), IAANSAD and SA Branch of ANS, 1998, 461–465.
18. Hahn, H. and Klier, W., 'Control of spatial parallel robots with redundant and non redundant servopneumatic actuators', 2000.

19. Hahn, H., Klier, W. and Leimbach, K.-D., 'Nonlinear control of planar parallel robots with redundant servopneumatic actuators', *Zeitschrift für angewandte Mathematik und Mechanik (ZAMM)* **79** (S1), 1999, 79–82.
20. Hahn, H. and Leimbach, K.-D., 'Control strategies of servo-hydraulic multi-axis test facilities', in *Proceedings of the 2nd International Symposium Environmental Testing for Space Programmes*, ESA/ESTEC, Noordwijk, The Netherlands, 1993, 77–84.
21. Hahn, H., Leimbach, K.-D. and Zhang, X., 'Theoretical modelling and control concept of a multi-axis servohydraulic test facility', in *Proceedings of the IFAC Symposium on Large Scale System (LLS)*, IFAC, Peking, B. Lin, T. Chen and Y.P. Zheng (eds), Chinese Association of Automation, 1992, 260–265.
22. Hahn, H. and Piepenbrink, A., 'Mathematical modelling, experimental identification and nonlinear control of a servopneumatic actuator, Part I', 1998, to appear.
23. Hecker, F., 'Identifikationsgestützte Regelung eines servopneumatischen Mehrachsenprüfstandes', Ph.D. Thesis, Fachgebiet Regelungstechnik und Systemdynamik (Maschinenbau), Universität-Gh Kassel, 1997.
24. Husty, M.L., 'An algorithm for solving the direct kinematic of Stewart–Gough-type platforms', *Mechanism and Machine Theory*, **31**(4), 1996, 365–379.
25. Juedes, D., 'A taxonomy of automatic differentiation tools', in *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, A. Griewank and G.F. Corliss (eds), SIAM, Philadelphia, PA, 1991, 315–329.
26. Kalaba, R. and Tischler, A., 'Automatic derivative evaluation in the optimization of nonlinear models', *The Review of Economics and Statistics* **66**, 1984, 653–660.
27. Kalman, D. and Lindell, R., 'Automatic differentiation in astrodynamical modeling', Aerospace Report ATR-92(8151)-1, The Aerospace Corporation, Engineering and Technology Group, El Segundo, CA, 1992.
28. Klier, W. and Hahn, H., 'Mathematisch-physikalische Modellgleichungen und Regelungsstrategien räumlicher servopneumatischer Parallelroboter', in *Tagungsband zur VDI/VDE-GMA Fachtagung Robotik 2000*, VDI-Verlag, Berlin, 2000, 71–76.
29. Klier, W., Hahn, H. and Neumann, M., 'Mathematical modeling, computer simulation and control concepts of spatial servopneumatic parallel robots', in *Proceedings of the 32nd International Symposium on Robotics (ISR 2001)*, Seoul, Korea, 2001, 1719–1724.
30. Lazard, D., *On the Representation of Rigid-Body Motions and Its Application to Generalized Platform Manipulators*, Kluwer Academic Publishers, Dordrecht, 1993, 175–182.,
31. Linnainmaa, S., 'Taylor expansion of the accumulated rounding error', *BIT (Nordisk Tidskrift for Informationsbehandling)*, **16**(1), 1976, 146–160.
32. Lorenc, A.C., 'A global three-dimensional multivariate statistical interpolation scheme', *Monthly Weather Review* **109**, 1981, 701–721.
33. Merlet, J.-P., 'Parallel manipulators: State of the art and perspectives', *Advanced Robotics* **8**, 1994, 589–596.
34. Raghavan, M., 'The Stewart platform of general geometry has 40 configurations', *Journal of Mechanical Design* **115**, 1993, 277–282.
35. Rall, L.B., *Automatic Differentiation: Techniques and Applications*, Lecture Notes in Computer Science, Vol. 120, Springer-Verlag, Berlin, 1981,
36. Röbenack, K., 'Nutzung des automatischen Differenzierens in der nichtlinearen Regelungstheorie', in *Tagungsband 2. Gemeinsamer GAMM-GMA-Workshop "Theoretische Verfahren der Regelungstechnik"*, Kassel, P.C. Müller (ed.), Bergische Universität-Gh Wuppertal, 1999.
37. Rothfuß, R. and Zeitz, M., 'A toolbox for symbolic nonlinear feedback design', in *Proceedings of 13th IFAC World Congress*, San Francisco, CA, J. Gertler, J. Gernz and M. Peskin (eds), Pergamon Press, 1995, 283–288.
38. Schiehlen, W., 'Computer generation of equations of motion', *Computer Aided Design and Optimization of Mechanical System Dynamics*, Springer-Verlag, Berlin, 1984, 183–215.

39. Schiehlen, W., 'Symbolic computations in multibody systems', *Computer-Aided Analysis of Rigid and Flexible Mechanical Systems* Kluwer Academic Publishers, Dordrecht, 1994, 101–136.
40. Schiehlen, W. and Kreuzer, E., 'Rechnergestütztes Aufstellen der Bewegungsgleichungen gewöhnlicher Mehrkörpersysteme', *Ingenieur-Archive* **46**, 1977, 185–195.
41. Schlacher, K., Scheidl, R., Meindl, W., Kicking, R. and Fuchs, W., 'Automatic derivation of symbolic state equations for mechatronic systems', in *Proceedings 2nd European Nonlinear Oscillations Conference*, Prague, Vol. 2, P. Pust (ed.), EUROMECH, 1996.
42. Sela, J.G., 'Spectral modeling at the national meteorological center', *Monthly Weather Review* **108**, 1980, 1279–1292.
43. Soulié, E.J., 'A few examples of least squares optimization in physical chemistry and astronomy', in *Trends in Mathematical Optimization*, K.-H. Hoffmann, J.B. Hiriart-Urruty, C. Lemaréchal and J. Zowe (eds), Birkhäuser Verlag, Basel, 1988, 327–340.
44. Speelpenning, B., 'Compiling fast partial derivatives of functions given by algorithms', Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, IL, 1980.
45. Stewart, D., 'A platform with six degrees of freedom', in *Proceedings of the Institute for Mechanical Engineering*, Institute for Mechanical Engineering, London, 1965, 371–386.