AN EFFICIENT METHOD TO COMPUTE LIE DERIVATIVES AND THE OBSERVABILITY MATRIX FOR NONLINEAR SYSTEMS

Klaus Röbenack† and Kurt J. Reinschke†

†*Institut für Regelungs- und Steuerungstheorie, TU Dresden, Mommsenstr. 13, D-01062 Dresden, Germany*
klaus@roebenack.de, kr@erss11.et.tu-dresden.de

Abstract— The design of a normal form observer requires the computation of the observability matrix. The authors propose a new approach for the computation of Lie derivatives and the observability matrix based on automatic differentiation.

I. INTRODUCTION

Many physical systems can be described by ordinary differential equations (ODEs)

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0 \tag{1}$$

with an output

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t)). \tag{2}$$

Let us assume that the maps $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ and $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^p$ are sufficiently smooth. In practical applications the dimension p of the output is usually much smaller than the dimension n of the state-space. Since most techniques for controller design require knowledge of the state (both conventional approaches [1–3] and chaos control techniques [4, 5]), the state $\mathbf{x}$ has to be reconstructed by observing the output $\mathbf{y}$. The most common way is to estimate the state by a second dynamical system called *observer*. A system (1,2) is called *observable* if every pair of different initial values is distinguishable with respect to the output signal [2]. Unfortunately, many systems are not observable due to an unfavourable output map $\mathbf{h}$. In practical applications one can usually chose between different sensors leading to different output maps. Therefore, it is important to check the observability of a system. To do this it is convenient to look at the map

$$\Theta : \mathbf{x} \mapsto \begin{pmatrix} \mathbf{y} \\ \dot{\mathbf{y}} \\ \vdots \\ \mathbf{y}^{(d)} \end{pmatrix}. \tag{3}$$

If the map Θ is invertible (i.e., injective) for a fixed d, it is possible to reconstruct the state from the output signal. It is basically impossible to check the global invertability for general nonlinear maps. However, the Implicit Function Theorem (or, in more general cases the Rank Theorem) provides a sufficient condition for local invertability. The map Θ is locally invertible at $\mathbf{x}_0$ if the Jacobian has full rank, i.e., if

$$rank \left(\left. \frac{d\Theta(\mathbf{x})}{d\mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} \right) = n. \tag{4}$$

Hence, the system is locally observable if the condition (4) holds. To compute this Jacobian (which is called *observability matrix*, see [6]) we have to compute the map Θ first. Let $L_\mathbf{f}\mathbf{h}(\mathbf{x})$ denote the Lie derivative of $\mathbf{h}$ along the vector field $\mathbf{f}$. Higher order Lie derivatives are defined by

$$L_\mathbf{f}^k\mathbf{h}(\mathbf{x}) := \frac{\partial L_\mathbf{f}^{k-1}\mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}) \quad \text{with} \quad L_\mathbf{f}^0\mathbf{h}(\mathbf{x}) := \mathbf{h}(\mathbf{x}).$$

With the chain rule we obtain the identity

$$\mathbf{y}^{(k)}(t) \equiv L_\mathbf{f}^k\mathbf{h}(\mathbf{x}(t)) \qquad (k \in \mathbb{N}) \tag{5}$$

and get a series representation of the output $\mathbf{y}$ (see [1, p. 140]):

$$\mathbf{y}(t) = \sum_{k=0}^{\infty} L_\mathbf{f}^k\mathbf{h}(x_0)\frac{t^k}{k!}$$

For small systems the Lie derivatives can be computed with computer algebra packages (e.g. MATHEMATICA, MAPLE). This approach has two drawbacks in case of large scale systems. First, for higher order derivatives the computational effort (computation time, memory requirements) can increase drastically due to chain, product and quotient rules [7, 8]. Furthermore, symbolic differentiation usually entails a repeated evaluation of common expressions. Second, the system (1) may not be described by an explicitly given mathematical function but by an algorithm containing conditional branches and loops. On the other hand, difference quotients like

$$\frac{F(x+h) - F(x)}{h} \quad \text{or} \quad \frac{F(x+h) - F(x-h)}{2h}$$

do not provide accurate values because of cancellation and truncation errors. For higher order derivatives these accuracy problems become acute. These disadvantages of symbolical and numerical differentiation can be circumvented with *automatic differentiation* [9]. Similar to the symbolic differentiation, elementary differentiation rules like sum, product and chain rule will be applied systematically. Tools for automatic differentiation will use concrete numbers instead of symbolic expressions.

II. AUTOMATIC DIFFERENTIATION

There are two basic concepts for the implementation of automatic differentiation software [9]: *program transformation* with compiler-generators and *operator overloading* in case of object oriented programming languages like C++ or Fortran90. Operator overloading can be employed to compute univariate Taylor series. Consider a smooth map $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^m$ which maps a curve $\mathbf{x}$ into a curve $\mathbf{z}$. Let $\mathbf{x}$ be given by

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{x}_1 t + \mathbf{x}_2 t^2 + \cdots + \mathbf{x}_d t^d \qquad (6)$$

with vector-valued coefficients $\mathbf{x}_0, \ldots, \mathbf{x}_d \in \mathbb{R}^n$. We will use the function $\mathbf{F}$ to map the curve $\mathbf{x}$ into a curve $\mathbf{z}$. Provided that $\mathbf{F}$ is d times continously differentiable, we can express $\mathbf{z}$ by the Taylor expansion

$$\mathbf{z}(t) = \mathbf{z}_0 + \mathbf{z}_1 t + \mathbf{z}_2 t^2 + \cdots + \mathbf{z}_d t^d + \mathcal{O}(t^{d+1}) \qquad (7)$$

with

$$\mathbf{z}_j = \frac{1}{j!} \frac{\partial^j \mathbf{z}(t)}{\partial t^j}\bigg|_{t=0}.$$

Each Taylor coefficient $\mathbf{z}_j \in \mathbb{R}^m$ is uniquely determined by the coefficients $\mathbf{x}_0, \ldots, \mathbf{x}_j$. In particular, we have

$$\begin{aligned}
\mathbf{z}_0 &= \mathbf{F}(\mathbf{x}_0) \\
\mathbf{z}_1 &= \mathbf{F}'(\mathbf{x}_0)\mathbf{x}_1 \\
\mathbf{z}_2 &= \mathbf{F}'(\mathbf{x}_0)\mathbf{x}_2 + \tfrac{1}{2}\mathbf{F}''(\mathbf{x}_0)\mathbf{x}_1\mathbf{x}_1 \\
\mathbf{z}_3 &= \mathbf{F}'(\mathbf{x}_0)\mathbf{x}_3 + \mathbf{F}''(\mathbf{x}_0)\mathbf{x}_1\mathbf{x}_2 + \tfrac{1}{6}\mathbf{F}'''(\mathbf{x}_0)\mathbf{x}_1\mathbf{x}_1\mathbf{x}_1
\end{aligned}$$

Using automatic differentiation, the Taylor coefficients of the output signal will be obtained without symbolic computations of the derivative tensors $\mathbf{F}', \mathbf{F}'', \ldots, \mathbf{F}^{(d)}$. One has only to overload elementary functions and operations. Some examples of the associated calculations are listed in Table 1. These rules are already implemented in Automatic Differentiation packages such as ADOL-C [10] and TADIFF [11] for C++, or AD01 [12] for FORTRAN90. This direct computation of Taylor coefficients is called *forward mode* of automatic differentiation.

In the *reverse mode* of automatic differentiation one can compute the matrices $A_0, \ldots, A_d \in \mathbb{R}^{m \times n}$ of the Jacobian path

$$\mathbf{F}'(\mathbf{x}(t)) = A_0 + A_1 t + A_2 t^2 + \cdots$$

The matrices $A_0, \ldots, A_d$ are partial derivatives of the Taylor coefficients of the curves $\mathbf{x}$ and $\mathbf{z}$, i.e., there holds the following identity [13]:

$$\frac{\partial \mathbf{z}_j}{\partial \mathbf{x}_i} = \frac{\partial \mathbf{z}_{j-i}}{\partial \mathbf{x}_0} = \begin{cases} A_{j-i} & \text{if } j \geq i \\ 0 & \text{otherwise} \end{cases}$$

Each matrix A_j is uniquely determined by the Taylor coefficients $\mathbf{x}_0, \ldots, \mathbf{x}_j$. The computation can be implemented in a very efficient way [9, 10]. With an single application of the reverse mode to the dth Taylor coefficient $\mathbf{z}_d = \mathbf{z}_d(\mathbf{x}_0, \ldots, \mathbf{x}_d)$ one gets all partial derivatives $A_0, \ldots, A_d$ at once.

III. COMPUTATION

Automatic differentiation can be employed to compute an series expansion of the autonomous ODE (1). Simultaneously, we will interpret $\mathbf{f}$ as a map between two curves $\mathbf{x}$ and $\mathbf{z}$, where $\mathbf{x}$ and $\mathbf{z}$ are given by the Taylor expansion (6) and (7). For known Taylor coefficients $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_i$ of $\mathbf{x}$, the Taylor coefficients $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_i$ of $\mathbf{z}$ can be computed with automatic differentiation. Because of the identity $\mathbf{z}(t) \equiv \dot{\mathbf{x}}(t)$, we can calculate the Taylor coefficient $\mathbf{x}_{i+1}$:

$$\mathbf{x}_{i+1} = \frac{1}{i+1}\,\mathbf{z}_i \qquad (8)$$

Beginning with a given initial value $\mathbf{x}_0$ one can easily expand the solution of the ODE (see [14]). Furthermore, we can compute the Taylor coefficients $\mathbf{y}_0, \ldots, \mathbf{y}_d \in \mathbb{R}^p$ of the output signal

$$\mathbf{y}(t) = \mathbf{y}_0 + \mathbf{y}_1 t + \mathbf{y}_2 t^2 + \cdots \qquad (9)$$

by automatic differentiation (see Fig 1). With this results we can express the function values of the Lie derivatives $L_{\mathbf{f}}^k \mathbf{h}(x_0)$ in terms of the Taylor coefficients calculated above (see [15]):

$$L_{\mathbf{f}}^k \mathbf{h}(\mathbf{x}_0) = k\,!\,\mathbf{y}_k$$

Next, we want to compute the Jacobians $\frac{\partial}{\partial \mathbf{x}} L_{\mathbf{f}}^k \mathbf{h}(\mathbf{x})$. To do this we take the variational equation

$$\dot{\tilde{x}}(t) = A(t)\,\tilde{x}(t), \quad \tilde{y}(t) = C(t)\,\tilde{x}(t) \qquad (10)$$

with $A(t) = \mathbf{f}'(\mathbf{x}(t))$ and $C(t) = \mathbf{h}'(\mathbf{x}(t))$ into consideration. The Taylor coefficients $A_i \in \mathbb{R}^{n \times n}$ and $C_i \in \mathbb{R}^{n \times p}$ of the matrices

$$\begin{aligned}
A(t) &= A_0 + A_1 t + A_2 t^2 + \cdots \\
C(t) &= C_0 + C_1 t + C_2 t^2 + \cdots
\end{aligned}$$

are partial derivatives between Taylor coefficients [13]:

$$A_i = \frac{\partial \mathbf{z}_i}{\partial \mathbf{x}_0}, \qquad C_i = \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_0}.$$

Operations			Taylor coefficients			
z	$=$	$x \pm y$	z_k	$=$	$x_k \pm y_k$	
z	$=$	$x \cdot y$	z_k	$=$	$\sum_{i=0}^{k} x_i \, y_{k-i}$	
z	$=$	e^x	z_k	$=$	$\frac{1}{k} \sum_{i=0}^{k-1} (k-i) \, z_i \, x_{k-i}$,	$k \geq 1$
z	$=$	$\ln(x)$	z_k	$=$	$\frac{1}{x_0} \left(x_k - \frac{1}{k} \sum_{i=1}^{k-1} i \, z_i \, x_{k-i} \right)$,	$k \geq 1$

Table 1: Computation of Taylor coefficients [11]

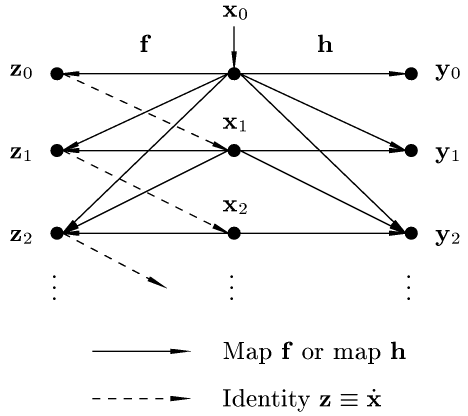


Figure 1: Signal flow for the computation of the Taylor coefficients $\mathbf{y}_0, \ldots, \mathbf{y}_d$

Applying the chain rule, we can get the total derivatives by the following calculations [14, 16]:

$$\begin{aligned} B_k &= \tfrac{1}{k+1} \tfrac{d\mathbf{z}_k}{d\mathbf{x}_0} \\ &= \tfrac{1}{k+1} \sum_{j=0}^{k} \tfrac{\partial \mathbf{z}_k}{\partial \mathbf{x}_j} \tfrac{d\mathbf{x}_j}{d\mathbf{x}_0} \\ &= \tfrac{1}{k+1} \left(A_k + \sum_{j=1}^{k} A_{k-j} B_{j-1} \right), \\ D_k &= \tfrac{d\mathbf{y}_k}{d\mathbf{x}_0} \\ &= \sum_{j=0}^{k} \tfrac{\partial \mathbf{y}_k}{\partial \mathbf{x}_j} \tfrac{d\mathbf{x}_j}{d\mathbf{x}_0} \\ &= C_k + \sum_{j=1}^{k} C_{k-j} B_{j-1}. \end{aligned}$$

Now we are able to describe the Jacobians in terms of Taylor coefficients

$$\left. \frac{\partial L_{\mathbf{f}}^k \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} = \left. \frac{d\mathbf{y}^{(k)}(t)}{d\mathbf{x}} \right|_{\substack{t=0 \\ \mathbf{x}=\mathbf{x}_0}} = k! \, \frac{d\mathbf{y}_k}{d\mathbf{x}_0} = k! \, D_k.$$

Hence, the observability matrix at $\mathbf{x}_0$ can be written as

$$\left. \frac{\partial \Theta(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} = \begin{pmatrix} 0! \cdot D_0 \\ \vdots \\ (n-1)! \cdot D_{n-1} \end{pmatrix}.$$

The rank condition (4) should be checked with a singular value decomposition. Moreover, it is possible to compare different output maps by means of numerical properties of the observability matrices such as the condition number or the smallest singular value. The example will illustrate this.

IV. EXAMPLE

Let us consider the ODE

$$\underbrace{\begin{aligned} \dot{x}_1 &= -x_2 + x_1 (1 - x_1^2 - x_2^2) \\ \dot{x}_2 &= x_1 + x_2 (1 - x_1^2 - x_2^2) \\ \dot{x}_3 &= -x_3 (x_1^2 + x_2^2) \end{aligned}}_{\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})} \qquad (11)$$

For the initial value $\mathbf{x}(0) = (1, 0, 1)^\top$ we have the solution

$$\mathbf{x}(t) = (\cos(t), \sin(t), \exp(-t))^\top. \qquad (12)$$

We consider the output functions

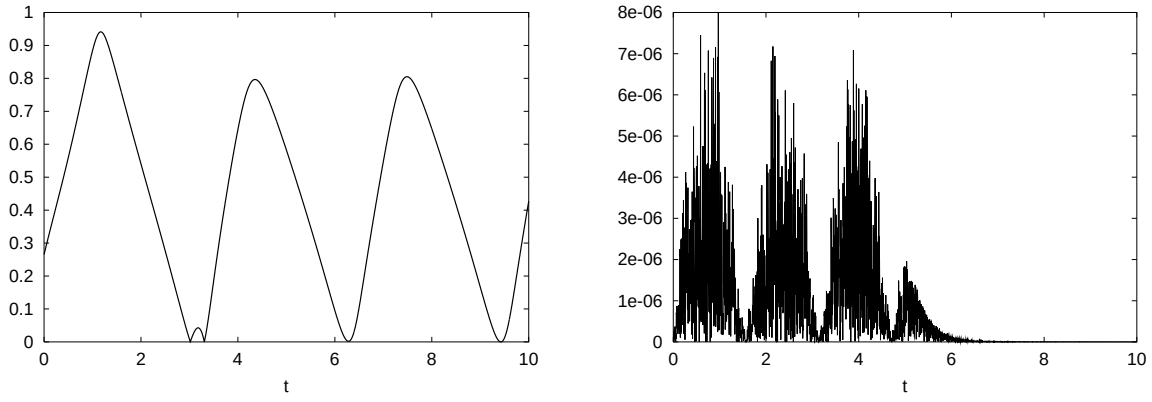
$$\begin{aligned} y_1 &= h_1(\mathbf{x}) = x_1 + x_2 + x_3 \quad \text{and} \\ y_2 &= h_2(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2. \end{aligned}$$

The smallest singular values of the observability matrices

$$\Theta_i(\mathbf{x}) := \begin{pmatrix} \nabla h_i(\mathbf{x}) \\ \nabla L_{\mathbf{f}} h_i(\mathbf{x}) \\ \nabla L_{\mathbf{f}}^2 h_i(\mathbf{x}) \end{pmatrix} \qquad (i = 1, 2)$$

along the solution (12) are given in Fig. 2.

The inversion of the map Θ_1 could cause problems only in some small areas (see Fig. 2(a)). The smallest singular value of the map Θ_2 takes values of a complete different range: All values are less than $8 \cdot 10^{-6}$. This suggests that the system (11) is close to be unobservable with the output map h_2.



(a) (b)

Figure 2: Smallest singular values of the observability matrices

References

[1] A. Isidori. *Nonlinear Control Systems: An Introduction*. Springer, 3. edition, 1995.

[2] E. D. Sontag. *Mathematical Control Theory*. Springer, 1990.

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

[4] E. Ott, C. Grebori, and J. A. Yorke. Controlling chaos. *Phys. Rev. Letters*, 64:1196–1199, 1990.

[5] M. J. Ogorzalek. *Chaos and complexity in nonlinear electronic circuits*, volume 22 of *World Scientific Series on Nonlinear Science, Series A*. World Scientific, 1997.

[6] R. Hermann and A. J. Krener. Nonlinear controllability and observability. *IEEE Trans. on Automatic Control*, AC-22(5):728–740, 1977.

[7] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pp. 83–108. Kluwer Academic Publishers, 1989.

[8] B. de Jager. The use of symbolic computation in nonlinear control: is it viable? *IEEE Trans. on Automatic Control*, AC-40(1):84–89, 1995.

[9] A. Griewank. *Evaluating Derivatives — Principles and Techniques of Algorithmic Differentiation*, volume 19 of *Frontiers in Applied Mathematics*. SIAM, 2000.

[10] A. Griewank, D. Juedes, and J. Utke. A package for automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Software*, 22:131–167, 1996. http://www.math.tu-dresden.de/wir/project/adolc/index.html.

[11] C. Bendtsen and O. Stauning. TADIFF, a flexible C++ package for automatic differentiation. Technical Report IMM-REP-1997-07, TU of Denmark, Dept. of Mathematical Modelling, Lungby, 1997.

[12] J. D. Pryce and J. K. Reid. AD01, a Fortran 90 code for automatic differentiation. Technical Report RAL-TR-1998-057, Rutherford Appleton Laboratory, Computing and Information Systems Department, 1998.

[13] B. Christianson. Reverse accumulation and accurate rounding error estimates for Taylor series. *Optimization Methods and Software*, 1:81–94, 1992.

[14] A. Griewank. ODE solving via automatic differentiation and rational prediction. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1995*, volume 344 of *Pitman Research Notes in Mathematics Series*. Addison-Wesley, 1995.

[15] K. Röbenack and K. J. Reinschke. Reglerentwurf mit Hilfe des Automatischen Differenzierens. *Automatisierungstechnik*, 48(2):60–66, 2000.

[16] K. Röbenack and K. J. Reinschke. Trajektorienplanung und Systeminversion mit Hilfe des Automatischen Differenzierens. In *Workshop des GMA-Ausschusses 1.4 "Neuere theoretische Verfahren der Regelungstechnik", Thun, Sept. 26-29*, pp. 232–242, 1999.