

# Nonlinear system identification employing automatic differentiation

Jan Schumann-Bischoff, Stefan Luther, Ulrich Parlitz

*Max Planck Institute for Dynamics and Self-Organization  
Am Faßberg 17, 37077 Göttingen, Germany*

*Institute for Nonlinear Dynamics, Georg-August-Universität Göttingen  
Am Faßberg 17, 37077 Göttingen, Germany*

*DZHK (German Center for Cardiovascular Research), partner site Göttingen,  
and Heart Research Center Göttingen, D-37077 Göttingen, Germany*

---

## Abstract

An optimization based state and parameter estimation method is presented where the required Jacobian matrix of the cost function is computed via automatic differentiation. Automatic differentiation evaluates the programming code of the cost function and provides exact values of the derivatives. In contrast to numerical differentiation it is not suffering from approximation errors and compared to symbolic differentiation it is more convenient to use, because no closed analytic expressions are required. Furthermore, we demonstrate how to generalize the parameter estimation scheme to delay differential equations, where estimating the delay time requires attention.

*Keywords:* nonlinear modelling, parameter estimation, delay differential equations, data assimilation

---

## 1. Introduction

For many processes in physics or other fields of science mathematical models exist (in terms of differential equations, for example), but not all state variables are easily accessible (measurable) and proper values of model parameters may be (partly) unknown. In particular, detailed biological cell models (e.g., cardiac myocytes [1]) may include many variables which are difficult to access experimentally and, in addition, depend on up to hundreds of physical parameters whose values have to be determined. To estimate unobserved variables (as a function of time) and model parameters different identification methods have been devised [2–14, 16–20]. These methods have in common that an attempt is made to adjust the model output (in general a function of the state variables) to some (experimentally) observed time series. To achieve agreement, unobserved variables and unknown model parameters are suitably adjusted such that the model reproduces and follows the observed time series. In geosciences and meteorology (e.g., whether forecasting) this procedure is often called *data assimilation* and describes the process of incorporating new (incoming) data into a computer model of the real system.

A general framework for state estimation provides, for example, the path integral formalism including a saddle point approximation [15, 16]. This formalism can be used to state the estimation problem as an optimization problem [12–14, 18, 19]. If an optimization method is employed that is based on gradient descent (such as the well-known Levenberg-Marquard method [21, 22]), in general the Jacobian matrix of the cost function has to be provided, whose derivation may be quite cumbersome (and error-prone), depending on the structure of the cost function and the underlying mathematical model of the dynamical system. To estimate the Jacobian matrix one may approximate it by numerical derivatives (often spoiled by unacceptably large truncation errors) or use symbolic mathematics, which requires, however, that the function to be derived has to be given in closed form.

A convenient alternative to both of these methods is *automatic differentiation* [24] where exact numerical values of the required derivatives are computed by analyzing a given source code implementation of the cost function. As

---

*Email addresses:* jan.schumann-bischoff@ds.mpg.de (Jan Schumann-Bischoff), stefan.luther@ds.mpg.de (Stefan Luther), ulrich.parlitz@ds.mpg.de (Ulrich Parlitz)

July 10, 2015

will be shown here automatic differentiation leads in this context not only to a very flexible and efficient algorithm for computing the required Jacobian but also provides the sparsity pattern of the Jacobian which is exploited by suitable optimization methods. In Section 2 we will give a formal description of the optimization problem to be solved for state and parameter estimation. Then we briefly present in Section 3 the concept of automatic differentiation in the form used here. As an illustrative example we show in Section 4 how to estimate the model parameters of the Lorenz-96 model. In Section 5 we discuss how to estimate the delay time in delay differential equations and provide in Section 6 an example (Mackey-Glass model).

## 2. State and parameter estimation method

The method used here to adapt a model to a time series is based on minimizing a cost function and was introduced in Ref. [19]. For completeness we present in the following an extended version covering also delay differential equations (DDEs).

We assume that a multivariate  $R$ -dimensional time series  $\{\boldsymbol{\eta}(n)\}$  is given consisting of  $N + 1$  samples  $\boldsymbol{\eta}(n) \hat{=} \boldsymbol{\eta}(t_n) \in \mathbb{R}^R$  measured at times  $\mathcal{T} = \{t_n = n \cdot \Delta t \mid n = 0, 1, \dots, N\}$ . For simplicity the observation times  $t_n$  are equally spaced (with a fixed time step  $\Delta t$ ) and start at  $t_0 = 0$ . The estimation method can easily be extended to nonuniformly sampled observations (see Ref. [19]). Here we consider the general case of a model given by a set of coupled delay differential equations (DDE)

$$\frac{d\mathbf{y}(t)}{dt} = \mathbf{F}(\mathbf{y}(t), \mathbf{y}_\tau(t), \mathbf{p}, t), \quad (1)$$

with  $\mathbf{y}_\tau(t) = \mathbf{y}(t - \tau)$ . The state vector(s)  $\mathbf{y}(t) = (y_1(t), \dots, y_D(t))^T$ , the delay parameter  $\tau \in \mathbb{R}$  and the  $U$  model parameters  $\mathbf{p} = (p_1, \dots, p_U)^T$  are unknown and have to be estimated from the time series  $\{\boldsymbol{\eta}(n)\}$ . Estimating  $\tau$  can not be conducted as estimating  $\mathbf{p}$ , because  $\mathbf{F}(\mathbf{y}(t), \mathbf{y}_\tau(t), \mathbf{p}, t)$  does *not* explicitly depend on  $\tau$ . In fact  $\mathbf{F}(\mathbf{y}(t), \mathbf{y}_\tau(t), \mathbf{p}, t)$  depends on  $\mathbf{y}_\tau(t)$  which is a function of  $\tau$ . We shall later come back to this topic.

Note that (1) also describes (as a special case) models given by coupled ordinary differential equations (ODEs). In this case the right-hand side of (1) is independent of  $\mathbf{y}_\tau(t)$  and thus can be replaced by  $\mathbf{F}(\mathbf{y}(t), \mathbf{p}, t)$  (see Ref. [19] for details).

To estimate the unknown quantities a measurement function

$$\mathbf{z}(t) = \mathbf{h}(\mathbf{y}(t), \mathbf{q}, t) \quad (2)$$

is required to represent the relation between model states  $\mathbf{y}(t)$  and the  $\mathbf{z}(t)$  corresponding to the observations  $\{\boldsymbol{\eta}(n)\}$ . This measurement function may contain  $V$  additional unknown parameters  $\mathbf{q} = (q_1, \dots, q_V)^T$  that also have to be estimated using information from the given time series  $\{\boldsymbol{\eta}(n)\}$ .

### 2.1. Cost function

The goal of the estimation process is to find a set of values for all unknown quantities such that the model equations provide via measurement function (2) a model times series  $\{\mathbf{z}(t_n)\}$  that matches the experimental time series  $\{\boldsymbol{\eta}(t_n)\}$ . In other words, the average difference between  $\boldsymbol{\eta}(t_n)$  and  $\mathbf{z}(t_n)$  should be small. Furthermore, the model equations should be fulfilled as well as possible. This means that modeling errors  $\mathbf{u}(t)$  are allowed, but should be small. Therefore, model (1) is extended to

$$\frac{d\mathbf{y}(t)}{dt} = \mathbf{F}(\mathbf{y}(t), \mathbf{y}_\tau(t), \mathbf{p}, t) + \mathbf{u}(t). \quad (3)$$

The smaller  $\mathbf{u}(t)$  is the better the model equations (1) are fulfilled. Next, for simplicity,  $\mathbf{u}(t)$  and  $\mathbf{y}(t)$  will be discretized at the times in  $\mathcal{T}$ . This means that  $\mathbf{y}(t)$  will be sampled at the same time when data are observed. With  $\mathbf{y}(n) \hat{=} \mathbf{y}(n \cdot \Delta t) \hat{=} \mathbf{y}(t_n)$  and  $\mathcal{Y}(a, b) = \{\mathbf{y}(n) \mid n = a, a + 1, \dots, b\}$  the set of values of the discretized model variables can be written as  $\mathcal{Y}(0, N)$ . The quantities in  $\mathcal{Y}(0, N)$  have to be estimated in addition to  $\mathbf{p}$  and  $\mathbf{q}$ . With the same discretization we have  $\{\mathbf{u}(n)\} = \{\mathbf{u}(t_n)\}$ . At this point we assume a fixed (not to be estimated) delay  $\tau = k \cdot \Delta t$  with  $k \in \mathbb{N}$  which is not necessarily equal to the delay parameter of the physical process underlying the data. This simplifies

the discretization of the delayed variable to  $\mathbf{y}_\tau(t) = \mathbf{y}(n \cdot \Delta t - k \cdot \Delta t) = \mathbf{y}((n-k) \cdot \Delta t) = \mathbf{y}(n-k) = \mathbf{y}_k(n)$ . The set of the discretized delayed variable is then  $\mathcal{Y}_k(0, N) = \mathcal{Y}(-k, N-k)$ . Note that  $\mathcal{Y}(-k, N-k) = \mathcal{Y}(-k, -1) \cup \mathcal{Y}(0, N-k)$ . Since  $\mathcal{Y}(0, N-k) \subset \mathcal{Y}(0, N)$ ,  $\mathcal{Y}(0, N-k)$  contains no additional quantities to be determined. Only the variables in  $\mathcal{Y}(-k, -1)$  are additional quantities which have to be estimated. Typically the delay time is much shorter than the length of the time series  $N \cdot \Delta t$  and hence the number of elements in  $\mathcal{Y}(-k, -1)$  is much smaller than in  $\mathcal{Y}(0, N)$ . Therefore the number of quantities to be estimated does not increase much compared to a model given by ODEs (with similar  $D$  and  $N$ ) where  $\mathcal{Y}(-k, -1)$  has not to be estimated.

The discretization of (3) is then given by

$$\mathbf{u}(n) \approx \left. \frac{\Delta \mathbf{y}}{\Delta t} \right|_{t_n} - \mathbf{F}(\mathbf{y}(n), \mathbf{y}_k(n), \mathbf{p}, t_n), \quad (4)$$

whereas the symbol  $\left. \frac{\Delta \mathbf{y}}{\Delta t} \right|_{t_n}$  stands for the finite difference approximation of  $\frac{d\mathbf{y}(t)}{dt}$  at time  $t_n$ . The goal of the adaption process is to minimize (on average) the norm of  $\mathbf{u}(n)$  and the norm of the difference  $\boldsymbol{\eta}(t_n) - \mathbf{z}(t_n)$ .

This leads to a cost function

$$C(\mathcal{Y}(-k, N), \mathbf{p}, \mathbf{q}) = C_1 + C_2 + C_3 + C_4 \quad (5)$$

with

$$C_1 = \frac{\alpha}{N} \cdot \sum_{n=0}^N (\boldsymbol{\eta}(n) - \mathbf{z}(n))^T \mathbf{A} (\boldsymbol{\eta}(n) - \mathbf{z}(n)) \quad (6)$$

$$C_2 = \frac{1-\alpha}{N} \cdot \sum_{n=0}^N \mathbf{u}(n)^T \mathbf{B} \mathbf{u}(n) \quad (7)$$

$$C_3 = \frac{1-\alpha}{N} \cdot \sum_{n=3}^{N-2} (\mathbf{y}_{\text{apr}}(n) - \mathbf{y}(n))^T \mathbf{E} (\mathbf{y}_{\text{apr}}(n) - \mathbf{y}(n)) \quad (8)$$

$$C_4 = \frac{\beta}{L} \cdot \mathbf{q}(\mathbf{w}, \mathbf{w}_1, \mathbf{w}_u)^T \cdot \mathbf{q}(\mathbf{w}, \mathbf{w}_1, \mathbf{w}_u). \quad (9)$$

$C_1$  penalizes the difference between  $\boldsymbol{\eta}(n)$  and  $\mathbf{z}(n)$  whereas  $C_2$  penalizes large magnitudes of  $\mathbf{u}(n)$ .  $\mathbf{A}$ ,  $\mathbf{B}$ , and  $\mathbf{E}$  are weight matrices that will be specified later. At the minimum of (5) the solution  $(\hat{\mathcal{Y}}(-k, N), \hat{\mathbf{p}}, \hat{\mathbf{q}})$  is obtained which is considered as the solution of the estimation problem. In the term  $C_3$  a Hermite interpolation is performed to determine  $\mathbf{y}_{\text{apr}}(n)$  from neighboring points and the time derivatives which are, according to (3), given by

$$\mathbf{G}(\mathbf{y}(t), \mathbf{y}_\tau(t), \mathbf{p}, t) = \mathbf{F}(\mathbf{y}(t), \mathbf{y}_\tau(t), \mathbf{p}, t) + \mathbf{u}(t). \quad (10)$$

With Eq. (10) the Hermite interpolation reads

$$\begin{aligned} \mathbf{y}_{\text{apr}}(n) = & \frac{11}{54} [\mathbf{y}(n-2) + \mathbf{y}(n+2)] + \frac{8}{27} [\mathbf{y}(n-1) + \mathbf{y}(n+1)] \\ & + \frac{\Delta t}{18} [\mathbf{G}(\mathbf{y}(n-2), \mathbf{y}_k(n-2), \mathbf{p}, t_{n-2}) - \mathbf{G}(\mathbf{y}(n+2), \mathbf{y}_k(n+2), \mathbf{p}, t_{n+2})] \\ & + \frac{4\Delta t}{9} [\mathbf{G}(\mathbf{y}(n-1), \mathbf{y}_k(n-1), \mathbf{p}, t_{n-1}) - \mathbf{G}(\mathbf{y}(n+1), \mathbf{y}_k(n+1), \mathbf{p}, t_{n+1})]. \end{aligned} \quad (11)$$

Smoothness of  $\mathcal{Y}(0, N)$  is enforced by small differences  $\mathbf{y}_{\text{apr}}(n) - \mathbf{y}(n)$ . The term  $C_3$  suppresses non-smooth (oscillating) solutions which may occur without this term in the cost function. Let

$$\begin{aligned} \mathbf{w} &= (\mathcal{Y}(0, N), \mathcal{Y}(-k, -1), \mathbf{p}, \mathbf{q}) \\ &= (\mathcal{Y}(-k, N), \mathbf{p}, \mathbf{q}) \\ &= (w_1, \dots, w_L) \end{aligned} \quad (12)$$

be a vector containing all quantities to be estimated <sup>1</sup>. Again, if the model is given by ODEs,  $\mathcal{Y}(-k, -1)$  does not occur in  $\mathbf{w}$ . Hence for ODEs we obtain

$$\mathbf{w} = (\mathcal{Y}(0, N), \mathbf{p}, \mathbf{q}). \quad (13)$$

To force  $\mathbf{w}$  to stay between the lower and upper bounds  $\mathbf{w}_l$  and  $\mathbf{w}_u$ , respectively,  $\mathbf{q}(\mathbf{w}, \mathbf{w}_l, \mathbf{w}_u) = (q_1, \dots, q_L)^T$  is defined as

$$q_i(w_i, w_{l,i}, w_{u,i}) = \begin{cases} w_{u,i} - w_i & \text{for } w_i \geq w_{u,i} \\ 0 & \text{for } w_{l,i} < w_i < w_{u,i} \\ w_{l,i} - w_i & \text{for } w_i \leq w_{l,i} \end{cases} \quad (14)$$

$q_i$  is zero if the value of  $w_i$  lies within its bounds. To enforce this, the positive parameter  $\beta$  is set to a large number, e.g.  $10^5$ . In this paper the matrices  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{E}$  are diagonal matrices. The diagonal elements can be used for an individual weighting.

The homotopy parameter  $\alpha$  can be used to adjust whether the solution should be close to data ( $\alpha \approx 1$ ) or have a smaller error in fulfilling the model equations (see Ref. [18]). In [20] a possible technique is described to find an optimal  $\alpha$ . Furthermore one might use continuation (see Ref. [18]) where  $\alpha$  is stepwise decreased. Starting with  $\alpha \approx 1$  results in a solution close to the data. Then,  $\alpha$  is slightly decreased and the previously obtained solution is used as an initial guess and the cost function is optimized again. This procedure is repeated until the value  $\alpha = 0.5$  is reached.

Note that the cost function can be written in the form

$$C(\mathbf{w}) = \sum_{j=1}^J H_j(\mathbf{w})^2 = \|\mathbf{H}(\mathbf{w})\|_2^2 \quad (15)$$

where  $\mathbf{H}(\mathbf{w})$  is a high dimensional vector valued function of the high dimensional vector  $\mathbf{w}$ . To optimize (15) we use an implementation of the Levenberg-Marquardt algorithm [21, 22] called `sparseLM` [23]. Although  $C(\mathbf{w})$  will be optimized, `sparseLM` requires  $\mathbf{H}(\mathbf{w})$  and the sparse Jacobian of  $\mathbf{H}(\mathbf{w})$  as input. In the next section we discuss how to derive the Jacobian and its sparsity structure.

### 3. Automatic differentiation

The technique used here to estimate the variables and the parameters of a model from time series is based on minimizing the cost function (5) which can be written in the form of Eq. (15). The Levenberg-Marquardt algorithm used to minimize this cost function needs the vector valued function  $\mathbf{H}(\mathbf{w})$  and its Jacobian  $\partial\mathbf{H}/\partial\mathbf{w}$  with the elements  $\partial H_j/\partial w_l$  as input. Remember that the Jacobian has a sparse structure, i.e. it has many elements which are always zero. To compute this sparse Jacobian there exist three different techniques: *numerical differentiation*, *symbolic differentiation* and *automatic differentiation* (symbolic differentiation includes differentiation by hand). These techniques have particular advantages and disadvantages. *Numerical differentiation* is easy to implement, but numerically not exact. Furthermore, the sparsity pattern can not be detected reliably. *Symbolic differentiation* is numerically exact, but the function to be differentiated has to be available as a single expression. Deriving the Jacobian by hand usually is very error prone. Symbolic differentiation tools may help at this point, however, a change in the cost function requires deriving a new Jacobian. As an alternative the concept of *automatic differentiation* can be used. It is easy to implement, numerically exact and the sparsity pattern of the Jacobian can be detected automatically. Only the source code of the cost function is required by the automatic differentiation tool. Using automatic differentiation requires additional computational resources.

After weighing up the pros and cons of the discussed methods for computing the Jacobian we came to the conclusion that the concept of automatic differentiation is the most suitable one. Automatic differentiation is used here in terms of the tool (library) ADOL-C [25, 26]. ADOL-C provides functions to derive the numerical values of the

<sup>1</sup>Here we assume that a fixed but arbitrary rule is used to order the elements of the sets  $\mathcal{Y}(0, N)$  and  $\mathcal{Y}(-k, -1)$  to define the elements of the vector  $\mathbf{w}$ .

| <i>Example:</i>         | Mackey-Glass model | 80-dim Lorenz96 system |
|-------------------------|--------------------|------------------------|
| <i>Section</i>          | 6                  | 4                      |
| <i>CPU time for:</i>    |                    |                        |
| cost function calls     | 1.07s              | 31.8s                  |
| Jacobian calls (ADOL-C) | 2.34s              | 155s                   |
| optimizer (sparseLM)    | 3.14s              | 21845s                 |

Table 1: CPU times needed for evaluation of the values of the sparse Jacobian compared to the CPU time needed by the optimization routine. For the Mackey-Glass model only the CPU time for  $\tau \in [2.3, 2.4]$  from the **second step** of estimating the delay time is shown. The time values were measured on a 1.3GHz Dual core computer with 3GB memory.

Jacobian  $\partial \mathbf{H} / \partial \mathbf{w}$  of the function  $\mathbf{H}(\mathbf{w})$ . Furthermore the sparsity pattern of  $\partial \mathbf{H} / \partial \mathbf{w}$  can be detected and the numerical values of the non-vanishing elements can be derived (this functionality requires the graph coloring package ColPack [27]).

We used the Python interface Pyadoc [31] to ADOL-C wrapping functions of the ADOL-C library to Python. The advantage of using the Python interface instead of the C interface is that the cost function can be coded directly in Python using Numpy [30] arrays. ADOL-C is based on operator overloading. This means, that for computing the Jacobian the function to be differentiated has to be available as source code, only, with an input  $\mathbf{w}$  and return  $\mathbf{H}(\mathbf{w})$ . For evaluating the cost function usually the elements of  $\mathbf{w}$  have a numerical data type (e.g. integer, float, ...).

Typically deriving the sparsity pattern takes much more time than computing the non-zero values. However, this is more or less negligible because the detection of the sparsity pattern only has to be performed once, whereas the computation of the values of the non-zero elements occurs several times (always when the Jacobian has to be computed for a certain input  $\mathbf{w}$ , usually in each iteration of a numerical minimization routine).

For the examples in Sections 4 and 6 the time needed by ADOL-C used for computing the Jacobian of Eq. (15) was compared with the time needed by the used minimization routine (sparseLM) and the results are shown in Tab. 1. It turned out that the time needed by ADOL-C is much shorter than the time needed by sparseLM, i.e. the time needed to evaluate the cost function is negligible. Hence using ADOL-C does not lead to a significant increase of CPU time needed to solve the estimation problem.

#### 4. Lorenz-96 model

As described in Section 2 the estimation method can be used to adapt a system of ODEs to a time series (without time delay). As our first example we use the Lorenz-96 model that was introduced by E. Lorenz in 1996 [28]. Here the  $D = 80$  dimensional system is used given by the set of ODEs

$$\frac{dy_i}{dt} = y_{i-1}(t) \cdot (y_{i+1}(t) - y_{i-2}(t)) - y_i(t) + p \quad (16)$$

whereas  $i = 1, \dots, D$  is a cyclic index. This means that for  $i = D$  it is  $y_{i+1} = y_1$ . For  $i = 1$  it is  $i - 1 = D$  and  $i - 2 = D - 1$ . To compare the results from the estimation process a twin experiment is performed. The time series  $\{\boldsymbol{\eta}(t_n)\}$  with  $t_n \in \{0, 0.01, 0.02, \dots, 10\}$  is generated by integrating a similar model

$$\frac{dx_i}{dt} = x_{i-1}(t) \cdot (x_{i+1}(t) - x_{i-2}(t)) - x_i(t) + 8.17 \quad (17)$$

with the same dimension and cyclic index and taking the solution to build a noisy  $R = D/2 = 40$  dimensional multivariate time series by “observing” every second model variable,

$$\boldsymbol{\eta}_{ts}(t_n) = (x_1(t_n), x_3(t_n), x_5(t_n), \dots, x_{D-1}(t_n)). \quad (18)$$

A common type of noise in experimentally observed time series is white noise which is given by normally distributed random numbers with a variance  $\sigma^2$  and a mean which is zero. Adding the noise to the clean time series we obtain a

noisy multivariate time series

$$\boldsymbol{\eta}(t_n) = \boldsymbol{\eta}_{\text{ts}}(t_n) + \mathbf{r}(t_n), \quad \mathbf{r}(t_n) = (r_1(t_n), r_2(t_n), \dots, r_R(t_n)) \quad \text{with} \quad r_i(t_n) \sim \mathcal{N}(0, \sigma^2) \quad (19)$$

which is typical for experiments with measurement noise. To quantify the power of the clean signal and the noise one can define the **signal-to-noise-ratio** (SNR in dB) for each time series as

$$\text{SNR}(\{\eta_i(t_n)\}) = 10 \cdot \log_{10} \left( \frac{\sum_{n=0}^{N+1} (\eta_{\text{ts},i}(t_n) - \bar{\eta}_{\text{ts},i})^2}{\sum_{n=0}^{N+1} r_i(t_n)^2} \right) \quad (20)$$

(the overbar denotes the mean). The smaller the SNR the more measurement noise is present.

In this example the observed time series is given by

$$\boldsymbol{\eta}(t_n) = (x_1(t_n), x_3(t_n), x_5(t_n), \dots, x_{D-1}(t_n)) + \mathbf{r}(0, 1.0) \quad (21)$$

with a mean of the SNR of  $\overline{\text{SNR}}(\{\eta(t_n)\}) = 1/R \cdot \sum_{i=1}^R \text{SNR}(\{\eta_i(t_n)\}) \approx 10.7\text{dB}$ . The measurement function is given by

$$\mathbf{h}(\mathbf{y}(t), t) = (y_1(t), y_3(t), y_5(t), \dots, y_{D-1}(t)). \quad (22)$$

Next, the model (16) is adapted to  $\{\boldsymbol{\eta}(t_n)\}$  and the model variables and the parameter are estimated from the time series. The weighting matrices of the cost function (5) were fixed to:  $\mathbf{A} = \mathbb{1}_{D/2}$  ( $D/2 \times D/2$  unity matrix),  $\mathbf{B} = \mathbb{1}_D$ ,  $\mathbf{E} = 10^5 \cdot \mathbb{1}_D$  and  $\beta = 10^5$ . As mentioned in Section 2.1, continuation with  $\alpha$  is used here. This means that  $\alpha$  was set to the following values: 0.9999, 0.999, 0.99, 0.9, 0.5. First, the cost function was optimized with  $\alpha = 0.9999$ , then the resulting solution was used as the initial guess for the next optimization of the cost function with  $\alpha = 0.999$ . This procedure was repeated until  $\alpha = 0.5$  is reached. The solution of the optimization problem with  $\alpha = 0.5$  was then considered as the solution of the estimation problem. The results of the estimated solution for the model variables are shown in Fig. 1.

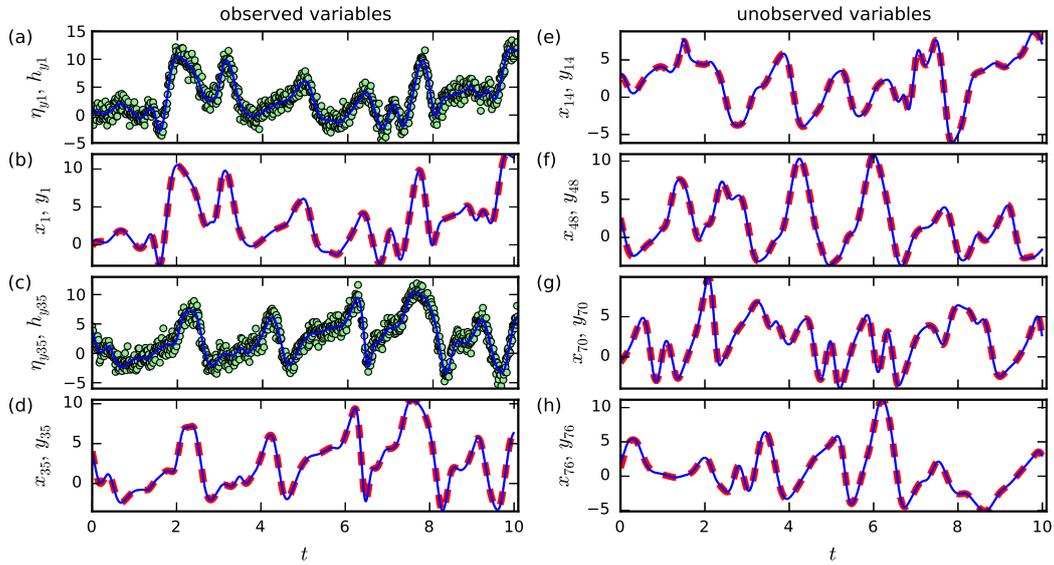


Figure 1: Adaptation of the 80 dimensional Lorenz96 model (Eq. (16)) to the time series  $\{\boldsymbol{\eta}(t_n)\}$  (Eq. (17), green circles). Every second model variable was observed. Left-hand side: The model output given by the measurement function  $\mathbf{h}(t)$  (Eq. (22), blue lines) was adapted to  $\{\boldsymbol{\eta}(t_n)\}$  ((a),(c)). (b) and (d) show the “true” solution  $x_i$  (red dashed lines) with the corresponding estimates  $y_i$  (blue lines), whereas for these quantities there exist measurements  $\eta_{y_i}$ . Right-hand side: (e), (f), (g) and (h) show estimates for the model variables  $y_i$  (blue lines) and the “true” solutions  $x_i$  (red dashed lines). Note that for all cases shown on the right-hand side there exist no measurements. The model parameter is estimated to  $p = 8.165$ , whereas 8.17 is used for generating the data by (17). Note that only a few of the 80 estimated model variables (with and without data) are shown.

One can see a good coincidence of the model variables  $\mathbf{y}(t)$  on the one hand and the “true” solution  $\mathbf{x}(t)$  on the other hand. This is also the case for the observations  $\{\boldsymbol{\eta}(t_n)\}$  and  $\mathbf{h}(\mathbf{y}(t))$ . Furthermore the modeling error  $\mathbf{u}(t)$  is small.

## 5. Delay differential equations and the estimation of delay parameters

An application where the cost function becomes rather complex and automatic differentiation turns out to be beneficial is the estimation of parameters and states of DDEs. How to derive the delayed variable  $\mathbf{y}_k(n) = \mathbf{y}(n - k)$  from  $\mathbf{y}(n)$  for  $\tau = k \cdot \Delta t$  with  $k \in \mathbb{N}$  was already discussed in Section 2.1. In this case  $\mathcal{Y}_k(0, N)$  can be computed from  $\mathcal{Y}(0, N)$  without interpolation. However, if  $k \in \mathbb{R}_+$  this approach does not work anymore. One has to interpolate  $\mathcal{Y}(0, N)$  to approximate  $\mathcal{Y}_k(0, N)$ . Because  $\Delta t$  will be small a linear interpolation should be sufficient and hence is used here.

In the general case we have  $\tau = (k' + l) \cdot \Delta t$  with  $\tau \in \mathbb{R}_+$ ,  $k' \in \mathbb{N}$  and  $l \in [0, 1[$ . Note that, due to these restrictions on  $k'$  and  $l$ ,  $k'$  and  $l$  are uniquely defined for a given  $\tau$ . Using this substitution for  $\tau$  the delayed variable can be written as

$$\begin{aligned} \mathbf{y}_k(n) &= \mathbf{y}_\tau(t_n) \\ &= \mathbf{y}(t_n - \tau) \\ &= \mathbf{y}(t_n - (k' + l) \cdot \Delta t) \\ &= \mathbf{y}(n \cdot \Delta t - (k' + l) \cdot \Delta t) \\ &= \mathbf{y}((n - k' + l) \cdot \Delta t) \\ &= \mathbf{y}(n - k' + l) \end{aligned} \tag{23}$$

and with linear interpolation between  $\mathbf{y}(n - k')$  and  $\mathbf{y}(n - k' + 1)$  we obtain

$$\begin{aligned} \mathbf{y}_k(n) &= \mathbf{y}(n - k' + l) \\ &= \mathbf{y}(n - k') + \frac{\mathbf{y}(n - k' + 1) - \mathbf{y}(n - k')}{\Delta t} \cdot l \cdot \Delta t \\ &= \mathbf{y}(n - k') + [\mathbf{y}(n - k' + 1) - \mathbf{y}(n - k')] \cdot l \end{aligned} \tag{24}$$

with  $\mathbf{y}_k(n) \in [\mathbf{y}(n - k'), \mathbf{y}(n - k' + 1)]$  for  $n = 0, 1, \dots, N$ . Note, that for  $l = 0$  we have  $\tau = k' \cdot \Delta t$  and hence the same situation as in Section 2.1, where  $\mathbf{y}_k(n) = \mathbf{y}(n - k')$  can be computed without an interpolation of the model variables.

One possibility to estimate  $\tau$  is discussed now. First,  $\tau$  must be added in Eq. (12) to the quantities to be estimated. Next, bounds for  $\tau$  must be set in Eq. (9). This means that there are bounds to  $k'$  so that (if the smallest bound of  $\tau$  is zero) we have  $k' \in [1, K]$ . If  $\tau$  and therefore  $k'$  and  $l$  would be fixed and not estimated,  $\mathbf{w}$  (Eq. (12)) would be  $\mathbf{w} = (\mathcal{Y}(-k', N), \mathbf{p}, \mathbf{q})$ . When  $\tau$  will be estimated, the number of elements in the history  $\mathcal{Y}(-k', -1)$  can vary during the optimization process due to variations in  $\tau$  (performed by the optimizer) and in  $k'$ . Because the number of elements of  $\mathbf{w}$  has to stay constant during one optimization process, the history with the largest possible number of elements, given by  $\mathcal{Y}(-K, -1)$ , must be added to  $\mathbf{w}$ , which then becomes  $\mathbf{w} = (\mathcal{Y}(-K, N), \mathbf{p}, \mathbf{q}, \tau)$ . Estimating  $\tau$  with this approach has one major disadvantage. When the optimizer evaluates the cost function with a certain  $\tau$ , first  $k'$  and  $l$  have to be derived from  $\tau$ . Next  $k'$  is used as an index to define the intervals  $[\mathbf{y}(n - k'), \mathbf{y}(n - k' + 1)]$  with  $n = 0, 1, \dots, N$  where to interpolate the model variables for deriving the delayed variables  $\mathbf{y}_k(n)$  with  $n = 0, 1, \dots, N$ . The cost function depends on the model equations, the model equations depend on the delayed variable and the delayed variable is, as shown in Eq. (24), a function of the model variables. Only the latter ones are quantities to be estimated. Hence the (sparse) Jacobian  $\partial H_i(\mathbf{w}) / \partial w_j$  of  $\mathbf{H}(\mathbf{w})$  (see Eq. (15)) depends on the (sparse) Jacobian  $\partial \mathbf{y}_k(n) / \partial \mathbf{y}(n)$  of the delayed variables to the model variables. For any specific  $n$  there exist  $K$  possible intervals  $[\mathbf{y}(n - \tilde{k}), \mathbf{y}(n - \tilde{k} + 1)]$  with  $\tilde{k} \in [1, K]$  where the linear interpolation might be performed although there is only the *one* interval with  $\tilde{k} = k'$  where it finally will be performed. The derivative of the linear interpolation to the model variable can be written as  $\partial \mathbf{y}_k(n) / \partial \mathbf{y}(n) = \partial \mathbf{y}_{(\tilde{k}+l)\Delta t}(n) / \partial \mathbf{y}(n)$  with  $\tilde{k} = 0, 1, \dots, N$ . Since only for  $\tilde{k} = k'$  a linear interpolation is performed, the relevant elements of the Jacobian matrix corresponding to  $\tilde{k} \neq k'$  are zero. Nevertheless, these elements which are zero at this step are not necessarily zero during the entire optimization process. Therefore these elements must be included in the sparsity pattern of nonzero elements, although, most of the time, they are zero. The sparsity pattern must not change during an optimization process. This fact would lead to a significant increase of (possible) nonzero elements in the Jacobian and hence would remove the advantage of dealing with a sparse Jacobian.

To avoid these problems the estimation of the delay parameter is divided into two steps:

**First step:** The cost function is minimized for several fixed delays,  $\tau = k \cdot \Delta t$  with  $k \in \mathbb{N}$ . In this case, as described in Section 2.1, no interpolation is necessary for computing the delayed variable. The solution of this step is  $\hat{C}(\tau)$ . This means that the assigned value is the value of the cost function in its minimum for a given  $\tau$ .  $\hat{C}(\tau)$  has a global minimum close to  $\tau_{\min} = k_{\min} \cdot \Delta t$ .

**Second step:** From the first step we have values for the cost function *only* around the minimum at  $\tau_{\min} - \Delta t$ ,  $\tau_{\min}$  and  $\tau_{\min} + \Delta t$ , as illustrated in Fig. 2. The global minimum at  $\hat{\tau}$  is either in the interval  $\hat{\tau} \in [\tau_{\min} - \Delta t, \tau_{\min}]$  or in the interval  $\hat{\tau} \in [\tau_{\min}, \tau_{\min} + \Delta t]$ . Now,  $\tau$  is estimated as well. As described at the beginning of this section, the delayed variable is computed by linear interpolation of the model variable. The cost function is minimized two times: once with  $\tau \in [\tau_{\min} - \Delta t, \tau_{\min}]$  and then with  $\tau \in [\tau_{\min}, \tau_{\min} + \Delta t]$ . The solution (model variables, parameters, delay time) obtained for the interval with the smallest value of the cost function after the optimization procedure is then taken as the final solution.

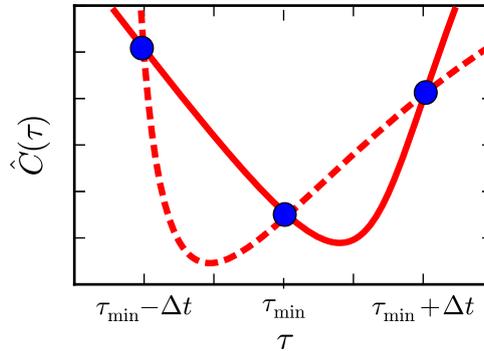


Figure 2: Estimation of the delay parameter: An illustration of the cost function  $\hat{C}(\tau)$  around the minimum at  $\tau_{\min}$  is shown. The values of  $\hat{C}(\tau)$  shown by the (blue) dots where obtained by the **first step** of estimating the delay parameter. The global minimum of  $\hat{C}(\tau)$  at  $\hat{\tau}$  is either in the interval  $\hat{\tau} \in [\tau_{\min} - \Delta t, \tau_{\min}]$  or in the interval  $\hat{\tau} \in [\tau_{\min}, \tau_{\min} + \Delta t]$ , as illustrated by the (red) dashed and continuous lines. Both lines describe possible behaviors of  $\hat{C}(\tau)$  between the (blue) dots. To determine whether the global minimum is in  $[\tau_{\min} - \Delta t, \tau_{\min}]$  or  $[\tau_{\min}, \tau_{\min} + \Delta t]$  both intervals are investigated separately in the **second step** using linear interpolation of the state variable.

## 6. Example: Mackey-Glass model

As an example of a DDE we consider the Mackey-Glass model [29]. This time delay model is given by

$$\frac{dy(t)}{dt} = p_1 \cdot \frac{y(t - \tau)}{1 + y(t - \tau)^{10}} - p_2 y(t) \quad (25)$$

and has the two model parameters  $p_1, p_2$  beside the delay parameter  $\tau$ . To generate a data time series a twin experiment is performed. This means that the model (similar to Eq. (25))

$$\frac{dx(t)}{dt} = 2 \cdot \frac{x(t - 2.38)}{1 + x(t - 2.38)^{10}} - 1 \cdot x(t) \quad (26)$$

was integrated first. The solution is then used to generate the (noisy) time series

$$\eta(t_n) = x(t_n) + \mathcal{N}(0, 0.1) \quad (27)$$

with  $t_n \in \{0, 0.1, \dots, 60\}$  and  $N = 601$  with timesteps  $\Delta t = 0.1$  (SNR = 8.0dB according to Eq. (20)).

Next the model (25) was adapted to  $\{\eta(t_n)\}$  using the measurement function

$$h(y(t), \bullet, \bullet) = y(t). \quad (28)$$

The parameters  $p_1, p_2$  and  $\tau$  were estimated in addition to the model variable. As described in Section 5 the estimation of  $\tau$  is divided into two steps:

|                              | $(p_1, p_2)$ | $\tau$ | $\hat{C}(\dots, \tau)$ |
|------------------------------|--------------|--------|------------------------|
| data                         | (2, 1)       | 2.38   |                        |
| estim. $\tau \in [2.2, 2.3]$ | (1.73, 0.86) | 2.300  | $5.41 \cdot 10^{-3}$   |
| estim. $\tau \in [2.3, 2.4]$ | (1.98, 0.99) | 2.387  | $4.99 \cdot 10^{-3}$   |

Table 2: Model parameters  $p_1$ ,  $p_2$  and delay parameter  $\tau$  of the Mackey-Glass model (25) estimated by adapting the model (25) to the time series given by Eq. (27). According to Fig. 3 the cost function  $\hat{C}(\hat{\mathcal{Y}}(-k, N), \hat{\mathbf{p}}, \bullet, \tau)$  has its global minimum either in  $\tau \in [2.2, 2.3]$  or in  $\tau \in [2.3, 2.4]$ . Minimizing the cost function and estimating  $\tau$  in addition to the model parameters and variables for both intervals results in a smaller  $\hat{C}(\hat{\mathcal{Y}}(-k, N), \hat{\mathbf{p}}, \bullet, \tau)$  for  $\tau \in [2.3, 2.4]$ .

- First  $\tau$  is fixed to several different values  $\tau = k \cdot \Delta t$  with  $k = 3, 4, \dots, 100$  and  $\Delta t = 0.1$ . For each fixed  $\tau$  the cost function is minimized. The value of the cost function at its minimum is denoted by  $\hat{C}(\hat{\mathcal{Y}}(-k, N), \hat{\mathbf{p}}, \bullet, \tau)$ . Its dependence on  $\tau$  is shown in Fig. 3. One can see a clear minimum at  $\tau = 2.3$ . Due to the step size of  $\Delta t = 0.1$  and the smoothness around this minimum one can expect that the global minimum is either at  $\tau \in [2.2, 2.3]$  or at  $\tau \in [2.3, 2.4]$ .
- Second the cost function is minimized two times (first with the bounds  $\tau \in [2.2, 2.3]$  and second with  $\tau \in [2.3, 2.4]$ ). In each case the delayed variable is approximated by linear interpolation according to Eq. (24). Remember that due to the bounds the index  $k'$  does *not* change during each optimization process and hence does not have to be recomputed from  $\tau$  in each iteration. Only  $l$  changes.

For all performed estimation processes the weighting matrices of the cost function (5) (in this cases they are scalar) and  $\alpha$  were fixed to:  $A = B = 1$ ,  $E = 10^3$ ,  $\beta = 10^5$  and  $\alpha = 0.5$ . The results for the estimated parameters and the delay parameter are shown in tab. 2.

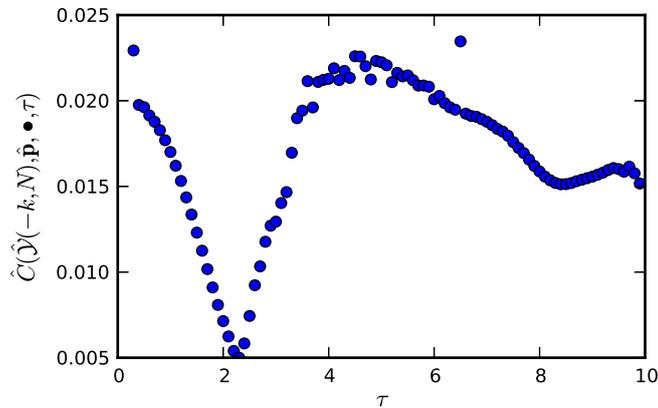


Figure 3: Cost function for adaption of the Mackey-Glass (25) model to a time series generated in a twin experiment with Eq. (27).

For  $\tau \in [2.3, 2.4]$  the cost function  $\hat{C}(\dots, \tau)$  at its minimum has a smaller value than for  $\tau \in [2.2, 2.3]$ . Hence the solution for  $\tau \in [2.3, 2.4]$  was chosen as the final result. Furthermore the estimated values for  $p_1$ ,  $p_2$  and  $\tau$  coincide much better with the “true” values used to generate  $\{\eta(t_n)\}$  (compared to the estimated values with  $\tau \in [2.2, 2.3]$ ). The estimated model variable for  $\tau \in [2.3, 2.4]$  is shown in Fig. 4. One can see a good coincidence of the model variable  $y(t)$  on the one hand and the “true” solution  $x(t)$  (Fig. 4b) and the data  $\{\eta(t_n)\}$  (Fig. 4a) on the other hand with a small modeling error  $u(t)$  (Fig. 4c).

## 7. Conclusion

Many optimization based system identification methods require information about derivatives of the underlying cost function in order to converge to the desired optimum. In general this information has to be provided by the user

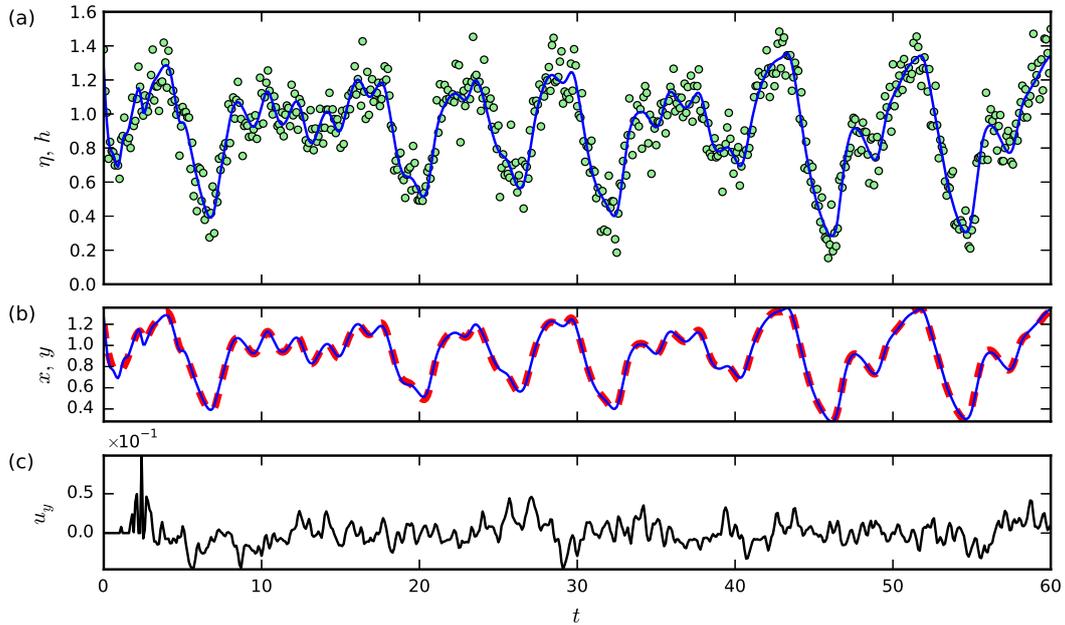


Figure 4: Adaption of the Mackey-Glass (25) model to the time series  $\eta$  (see (27) green circles) which was generated by the original “true” solution  $x$  (red dashed line, unknown to the estimation algorithm) using the measurement function  $h(y(t), \bullet, \bullet)$  (blue line, see (28)). The estimated solution for the model variable is  $y$  (blue line in (b)) and  $u$  (black line in (c)) is the error in the approximation of the model equation (see (4)). The delay time  $\tau \in [2.3, 2.4]$  was also estimated beside the model parameters and all estimated values are shown in tab. 2.

(or programmer) in terms of a Jacobian matrix, for example. Our examples show that this (typically cumbersome) task can be conveniently handled by means of automatic differentiation. This versatile tool from applied mathematics and computer science not only gives exact numerical values of the required derivatives, but also provides (and respects) the sparsity structure of the Jacobian matrix which may be exploited by any calling algorithm. These features have been demonstrated with a particular optimization algorithm (SparseLM) that enabled parameter and state estimation of the high dimensional Lorenz-96 system and the Mackey-Glass delay differential equation. A challenge for future research will be, for example, a successful application of the proposed parameter estimation to electrophysiological models of cardiac myocytes.

Our implementation (Python, C) of the estimation algorithm and a Python wrapper for sparseLM are available for download [32]. Other possible fields of application in nonlinear dynamics are bifurcation analysis and the computation of Lyapunov exponents and (covariant) Lyapunov vectors [33].

## Acknowledgements

The research leading to the results has received funding from the European Community’s Seventh Framework Program FP7/2007-2013 under grant agreement no. HEALTH-F2-2009-241526, EUTrigTreat. This work received support through Deutsche Forschungsgemeinschaft (SFB 1002 Modulatorische Einheiten bei Herzinsuffizienz). This work was supported by the DZHK (Deutsches Zentrum für Herz-Kreislauf-Forschung - German Centre for Cardiovascular Research). We thank H.D.I. Abarbanel and J. Bröcker for interesting and inspiring discussions on state and parameter estimation.

## Appendix A. Python example for automatic differentiation using ADOL-C

For illustration we present and discuss a Python example showing how to derive the sparse Jacobian of a function  $H(\mathbf{w})$  in Listing 1.

Listing 1: Simple Python example demonstrating the derivation of the (sparse) Jacobian of the function  $H(w)$  using the automatic differentiation tool ADOL-C.

```

1 import numpy as np
2 import adolc
3
4 def H(w):
5     h = np.zeros(3, dtype=w.dtype)
6     h[0] = 3*w[0]**2 + w[1]*w[3]
7     h[1] = 4*w[2]**3
8     h[2] = 5*w[0] + 2*np.exp(np.sin(w[1]*w[2]))
9     return h
10
11 w0 = np.array([4., 2., 3., 1.])
12
13 # Trace H(w)
14 # -----
15 adolc.trace_on(0)
16 aw = adolc.adouble(w0)
17 adolc.independent(aw)
18
19 Hfunc = H(aw)
20
21 adolc.dependent(Hfunc)
22 adolc.trace_off()
23
24 # Evaluate function jacobian at this point
25 w = np.array([-2., 3., 1., -4.])
26
27 # Evaluate function
28 valH = adolc.function(0,w)
29
30 # Evaluate dense jacobian
31 jacH = adolc.jacobian(0,w)
32 # Out jacH:
33 # [[-12. -4.  0.  3.]
34 # [ 0.  0.  12.  0.]
35 # [ 5. -2.280 -6.840 0.]]
36
37 # Derive sparsity pattern and evaluate values of nonzeros at w0
38 # -----
39 options = [0, 0, 0, 0]
40 [nnz, rind, cind, val] = adolc.colpack.sparse_jac_no_repeat(0,w0,options)
41 # Out nnz (number non-zeros): 7
42 # Out rind (row indices):
43 # [0 0 0 1 2 2 2]
44 # Out cind (column indices):
45 # [0 1 3 2 0 1 2]
46 # Out val (values):
47 # [ 24.  1.  2. 108.  5.  4.356  2.904]
48
49 # Evaluate values of nonzeros at w using nnz,
50 # rind, cind and valH from the previous step
51 # -----
52 [nnz, rind, cind, val] = adolc.colpack.sparse_jac_repeat(0,w,nnz, rind, cind, val)
53 # Out nnz (number non-zeros): 7
54 # Out rind (row indices):
55 # [0 0 0 1 2 2 2]
56 # Out cind (column indices):
57 # [0 1 3 2 0 1 2]
58 # Out val (values):
59 # [-12. -4.  3. 12.  5. -2.280 -6.840]

```

Before ADOLC can compute the numerical values of the Jacobian data type an internal function representation of  $H(w)$ , called trace, is created in lines 15-22. To do this the data type of the elements of the input vector is changed to the ADOL-C data type 'adouble' (line 16). Many (mathematical) functions are overloaded for this data type and can hence be used in the function to be differentiated. FOR-loops, WHILE-loops, IF ... THEN statements, etc. are also allowed under certain conditions. In lines 28 and 31 the function  $H(w)$  and its dense Jacobian are computed using the previously created trace for a new  $w$ . In line 40 the sparsity pattern of the Jacobian is detected and used in line 52 to compute the numerical values of its non-zero elements. The output in lines 33-35 and lines 55-59 show that the computed dense and sparse Jacobians are equal for the same  $w$ .

## References

- [1] R.H. Clayton, O. Bernus, E.M. Cherry, H. Dierckx, F.H. Fenton, L. Mirabella, S.V. Panfilov, F.B. Sachse, G. Seeman, H. Zhang, Models of cardiac tissue electrophysiology: Progress, challenges and open questions, *Prog. Biophys. Mol. Biol.* 104, 22-48 (2011).
- [2] U. Parlitz, L. Junge and L. Kocarev, Synchronization based parameter estimation from time series, *Phys. Rev. E* 54, 6253-6529 (1996).
- [3] H. U. Voss, J. Timmer and J. Kurths, Nonlinear dynamical system identification from uncertain and indirect measurements, *Int. J. Bifurcation and Chaos* 14, 1905-1933 (2004)
- [4] D. Ghosh and S. Banerjee, Adaptive scheme for synchronization-based multiparameter estimation from a single chaotic time series and its applications, *Phys. Rev. E* 78, 056211 (2008).
- [5] H. D. I. Abarbanel, D.R. Creveling, J.M. Jeanne, Estimation of parameters in nonlinear systems using balanced synchronization, *Phys. Rev. E* 77, 016208 (2008).
- [6] F. Sorrentino and E. Ott, Using synchronization of chaos to identify the dynamics of unknown systems *Chaos* 19, 033108 (2009).
- [7] I. G. Szendro, M. A. Rodríguez and J. M. López, On the problem of data assimilation by means of synchronization, *J. Geophys. Res.* 114, D20109 (2009).
- [8] W. Yu, G. Chen, J. Cao, J. Lü and U. Parlitz, Parameter identification of dynamical systems from time series, *Phys. Rev. E* 75, 067201 (2007).
- [9] D. Yu and U. Parlitz, Estimating parameters by autosynchronization with dynamic restrictions, *Phys. Rev. E* 77, 066221 (2008).
- [10] D. Ghosh, Nonlinear-observerbased synchronization scheme for multiparameter estimation, *EPL* 84, 40012 (2008).
- [11] D. Fairhurst, I. Tyukin, H. Nijmeijer and C. van Leeuwen, Observers for Canonic Models of Neural Oscillators, *Math. Model. Nat. Phenom.* 5(3), 146-185 (2010).
- [12] D.R. Creveling, P.E. Gill, H. D. I. Abarbanel, State and parameter estimation in nonlinear systems as an optimal tracking problem, *Phys. Lett. A* 372, 2640-2644 (2008).
- [13] J.C. Quinn, P.H. Bryant, D. R. Creveling, S. R. Klein, and H. D. I. Abarbanel, Parameter and state estimation of experimental chaotic systems using synchronization, *Phys. Rev. E* 80, 016201 (2009).
- [14] H. D. I. Abarbanel, D. R. Creveling, R. Farsian and M. Kostuk, Dynamical State and Parameter Estimation, *SIAM J. Appl. Dyn. Syst.* 8, 1341-1381 (2009).
- [15] H.D.I. Abarbanel, Effective actions for statistical data assimilation, *Phys. Lett. A* 373, 4044404 (2009).
- [16] J.C. Quinn, H.D.I. Abarbanel, State and parameter estimation using Monte Carlo evaluation of path integrals, *Q. J. R. Meteorol. Soc.* 136 (652), 1855-1867 (2010).
- [17] H.D.I. Abarbanel, M. Kostuk and W. Whartenby, Data assimilation with regularized nonlinear instabilities, *Q. J. R. Meteorol. Soc.* 136 (648), 769- 783 (2010).
- [18] J. Bröcker, On Variational Data Assimilation in Continuous Time, *Q. J. R. Meteorol. Soc.* 136, 1906-1919 (2010).
- [19] J. Schumann-Bischoff and U. Parlitz, State and parameter estimation using unconstrained optimization, *Phys. Rev. E* 84, 056214 (2011).
- [20] J. Bröcker and I. Szendro Sensitivity And Out-Of-Sample Error of Continuous Time Data Assimilation, *Q. J. R. Meteorol. Soc.* 138, 785-801 (2012).
- [21] K. Levenberg, A Method for the Solution of Certain Problems in Least Squares, *Quart. Appl. Math.* 2, 164-168 (1944).
- [22] D. Marquardt, An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *SIAM J. Appl. Math.* 11, 431-441 (1963).
- [23] M. I. A. Lourakis, Sparse Non-linear Least Squares Optimization for Geometric Vision, European Conference on Computer Vision 2, 43-56 (2010). [http://dx.doi.org/10.1007/978-3-642-15552-9\\_4](http://dx.doi.org/10.1007/978-3-642-15552-9_4) and <http://www.ics.forth.gr/~lourakis/sparseLM/>
- [24] L. B. Rall, Automatic Differentiation: Techniques and Applications, *Lecture Notes in Computer Science* 120 (1981).
- [25] A. Walther and A. Griewank, Getting started with ADOL-C, in U. Naumann and O. Schenk, *Combinatorial Scientific Computing*, Chapman-Hall CRC Computational Science, 181-202 (2012).
- [26] A. Griewank, D. Juedes and J. Utke, Algorithm 755: ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/ C++, *ACM Trans. on Math. Softw.* 22, 131-167 (1996).
- [27] ColPack: <http://www.cscapes.org/coloringpage/software.htm> (accessed: October 16, 2012)
- [28] E. N. Lorenz, Predictability – A problem partly solved, ECMWF, Reading, Berkshire, UK, Proceedings of the Seminar on Predictability 1, 1-18 (1996).
- [29] M. C. Mackey and L. Glass, Oscillation and chaos in physiological control systems, *Science* 197, 287-289 (1977).
- [30] Numpy: <http://numpy.scipy.org/> (accessed: October 16, 2012)
- [31] Pyadolc: <https://github.com/b45ch1/pyadolc> (accessed: October 16, 2012)
- [32] A Python wrapper for sparseLM and a Python-C implementation of the estimation method for ODEs can be downloaded from <http://www.bmp.ds.mpg.de/pysparselm.html> and <http://www.bmp.ds.mpg.de/pyodefit.html> (Gnu General Public License Version 3, 29 June 2007)
- [33] P.V. Kuptsov and U. Parlitz, Theory and Computation of Covariant Lyapunov Vectors, *J. Nonlinear Sci.* 22, 727-762 (2012).