NUMERICAL COMPUTATION OF NORMAL FORM COEFFICIENTS OF BIFURCATIONS OF ODES IN MATLAB

VIRGINIE DE WITTE AND WILLY GOVAERTS
Department of Applied Mathematics and Computer Science
Ghent University
Krijgslaan 281-S9, B-9000 Gent, Belgium

ABSTRACT. Normal form coefficients of codim-1 and codim-2 bifurcations of equilibria of ODEs are important since their sign and size determine the bifurcation scenario near the bifurcation points. Multilinear forms with derivatives up to the fifth order are needed in these coefficients. So far, in the MATLAB bifurcation software MatCont for ODEs, these derivatives are computed either by finite differences or by symbolic differentiation. However, both approaches have disadvantages. Finite differences do not usually have the required accuracy and for symbolic differentiation the MATLAB Symbolic Toolbox is needed. Automatic differentiation is an alternative since this technique is as accurate as symbolic derivatives and no extra software is needed. In this paper, we discuss the pros and cons of these three kinds of differentiation in a specific context by the use of several examples.

1. Introduction. The computation of the normal form coefficients of equilibrium bifurcations of dynamical systems requires multilinear forms of order up to five in order to compute directional derivatives. Derivatives of first or second order can be approximated by finite difference methods reasonably well if sophisticated heuristics are used; for higher order derivatives this is a hopeless task. Calculation of the multilinear forms by use of symbolic derivatives (SD) is not always possible in MATLAB since the Symbolic Toolbox is not always available. A third option for the computation of the directional derivatives is automatic differentiation (AD).

Every function $f$ expressible by a computer program is built from a finite set of elementary functions (this includes the basic arithmetic operations). The basic principle of automatic differentiation, see Griewank [10] and Griewank and Walther [11], is to use the known formulae for differentiating elementary functions together with the chain rule, to build up the needed derivatives of an arbitrary function.

We assume $f$ is a vector function $y = f(x)$ with $n$ inputs, or independent variables, $x = (x_1, \ldots, x_n)$ and $m$ outputs $y = (y_1, \ldots, y_m)$. The code for $f$ may contain branches and loops. However, each evaluation of $f$ at given inputs $x$ can be written as a code list, which is a finite sequence of assignments of the simple form

$$v_i = e_i(\text{previously defined } v_j \text{'s, or constants}), \quad i = 1, 2, \ldots, l+m, \quad (1)$$

where each $e_i$ is one of the elementary functions. The $v_i$ are called variables. In (1) it is convenient to use $v_{1-n}, \ldots, v_0$ as aliases for the inputs $x_1, \ldots, x_n$ and

2000 Mathematics Subject Classification. 37G05, 65D25, 65P30.
Key words and phrases. Automatic differentiation, symbolic derivatives, MatCont, directional derivatives.
\[ v_{l+1}, \ldots, v_{l+m} \] as aliases for the outputs \( y_1, \ldots, y_m \), following the notation of [10]. The remaining variables \( v_1, \ldots, v_l \) are called intermediate.

The forward mode of automatic differentiation is the simplest, and is appropriate to our application. Each \( v_i \) is represented by an object \( v_i \) of a data type that holds not just the value but some needed set of derivatives. For our purposes:

- There is (at any point) just one variable being treated as independent: we call it \( t \). Thus each variable \( v_i \) is regarded as a function of \( t \).
- The data structure holds some Taylor coefficients, namely the coefficients of the truncated Taylor series of \( v_i \), up to some order \( p \), expanded about some point \( t = a \). When we change to a new independent variable \( s = t - a \), we get that \( v_i \) holds \( (v_{i,0}, v_{i,1}, \ldots, v_{i,p}) \) where
  \[
  v(a + s) = v_{i,0} + v_{i,1}s + \cdots + v_{i,p}s^p + O(s^{p+1}).
  \]

We call the data type adtayl. It is helpful to think of an adtayl object as representing an infinite power series which is only known up to the order \( p \) term.

Details about the implementation are provided in [13]; here we just give a few basic ideas. When evaluating a code list, each elementary operation on real arguments is replaced by the corresponding operation on adtayl arguments. Consider first the four basic arithmetic operations. Suppose that \( a \) holds \((a_0, a_1, \ldots)\), and so on for other named variables. Let \( a \) be defined to order \( p \), and \( b \) to order \( q \). Then \( c = a + b \) and \( d = a \times b \) are defined to order \( r = \min(p, q) \) by

\[
\begin{align*}
  c_i &= a_i + b_i, \\
  d_i &= a_0b_i + a_1b_{i-1} + \cdots + a_nb_0,
\end{align*}
\]

and similarly for \( a - b \), and for \( a \div b \) provided \( b_0 \neq 0 \). For applying the standard functions \( \exp, \cos, \ldots \) to power series, there are various formulas in the literature.

We have aimed to choose ones that can be made reasonably fast in MATLAB, especially when the argument is a vector of power series, not just a single one. We give a few examples.

First, multiplication of power series is a convolution, which can be realised by the very fast built-in filter function of MATLAB.

Second, the method for \( \exp(a) \) exploits the fact that if \( c(t) = \exp(a(t)) \) then \( c'(t) = a'(t)c(t) \), which is converted to the integral form \( c(t) = c_0 + \int_0^t a'(s)c(s)ds \) where \( c_0 = e^{a_0} \). In terms of the coefficients this reduces to a triangular linear system, which is fast in MATLAB.

Third, for \( \sin(a) \) and \( \cos(a) \), it is convenient to compute both simultaneously as the real and imaginary parts of \( \exp(ia) \), and to record both results, along with the argument \( a \), in persistent storage. Each time a new \( a \) is given, it is checked against the recorded one. If they are equal, the result can be returned at once. Since \( \cos \) and \( \sin \) at the same argument often occur together in applications, this reduces the cost for these functions by nearly half.

Experimentally, we found in our MATLAB implementation that the run-time for computing a Taylor series up to order \( p \) is approximately proportional to \( p^{2.7} \). Since this involves the computation of all coefficients up to order \( p \), this is compatible with the known fact (see e.g. [11], Section 13.1) that the complexity of computing the \( d \)-th coefficient is proportional to \( d^2 \).

Regarding previous work to provide AD facilities in MATLAB, Rich and Hill [14] worked out a limited facility that enabled AD of simple expressions defined by a character string. The first significant work was that of Coleman and Verma
Table 1. Maximum order of multilinear forms in normal form coefficients for equilibrium bifurcations.

| bifurcation               | abbreviation | maximum order |
|---------------------------|--------------|---------------|
| Limit Point               | LP           | 2             |
| Bogdanov-Takens point     | BT           | 2             |
| Hopf point                | H            | 3             |
| Zero-Hopf point           | ZH           | 3             |
| Cusp point                | CP           | 3             |
| Generalized Hopf point    | GH           | 5             |
| Hopf-Hopf point           | HH           | 5             |

[3, 4], who produced an operator-overloading AD package ADMAT with facilities for forward and reverse mode for first and second derivatives, and run-time Jacobian sparsity detection. These authors interfaced ADMAT with ADMIT [5], a package for efficient sparse Jacobian calculation, and recently the ADiMat hybrid source-transformation/operator-overloading AD tool [15] has been developed. Comparisons [2] have shown its forward mode to be more efficient than that of ADMAT. Bischof et al. [1] proposed a novel software tool which transforms a given MATLAB program into another MATLAB program to change the semantics of the program in a fashion based on automatic differentiation. The crucial ingredient of the tool is a combination of source-to-source transformation and operator overloading. For first derivatives, currently the most efficient and comprehensive tool is probably Forth’s MAD package [8]. However, none of these tools handles Taylor series, although the subject is treated in [11], Chapter 13.

MatCont is a MATLAB numerical bifurcation toolbox for ODEs [6, 7]. So far, AD is not provided in MatCont. But it was already introduced in CL_MatContM, see [13], the pendant to MatCont for iterated maps. However, in CL_MatContM the computations of the multilinear forms \( A(q_1), B(q_1, q_2), C(q_1, q_2, q_3), D(q_1, q_2, q_3, q_4), E(q_1, q_2, q_3, q_4, q_5) \), where \( A \) is the Jacobian, \( B \) the tensor of second derivatives etc., were only implemented in the case of real vectors \( q_i \). We have extended this to the case of complex vectors. In [13] is shown that SD and AD have the same accuracy. Moreover, in the case of iterated maps, AD is faster for high values of the iteration number \( J \). When using AD, the elapsed time grows linearly with the iteration number \( J \), but for SD it grows more like \( J^d \) for \( d \)-th derivatives.

In Section 2 we give an overview of the maximal order of derivatives needed in the computation of the normal form coefficients of equilibrium bifurcations of codimension 1 and 2 and we motivate our choice of examples. In Section 3 we investigate three examples to demonstrate the advantages and disadvantages of the three kinds of derivatives. We give a comparison of time complexity and accuracy using finite differences (FD), symbolic derivatives and automatic differentiation. In Section 4 we draw some conclusions.

2. Preliminary remarks. In Table 1 we list all codimension 1 and 2 bifurcations of equilibria of ODEs and the corresponding maximum order of derivative appearing in the normal form coefficient. The most complicated normal form coefficients are those of the Generalized Hopf points and the Hopf-Hopf points, since these computations require the evaluation of several multilinear forms of order 4 and 5.
Therefore, we will consider in particular the AB-neuron model and the Morris Lecar example, which contain several of these bifurcation points.

We note that the Jacobian is needed during the continuation of a curve. AD could also be used for these computations. However, a very accurate Jacobian is not needed since errors in the Jacobian can be remedied by the Newton corrections. Therefore, in this case it is preferable to avoid AD since in our particular implementation AD is more time-consuming, as we will see later on, and there is a huge amount of Jacobian computations during the continuation. This would slow down the speed of the continuation unnecessarily. So the options of SD and FD are in this case sufficient.

Symmetry is not taken into account for the computation of the multilinear forms with either SD or AD. In the case of SD, we note that the speed of MATLAB depends strongly on the way in which the program deals with vectors. MATLAB is designed to perform vector and matrix operations efficiently. By making use of the symmetry this advantage would be lost. When using AD, the matrix of derivatives itself is never computed but only the product with certain vectors. Therefore, it is not quite clear how symmetry could be dealt with (see also [11], Section 13.3).

3. Test Cases. In this section we compare SD, AD and FD in a number of test cases. In the case of FD a multilinear form of order \( n \) is computed by a symmetric finite difference formula with \( O(h^2) \) accuracy that uses function evaluations in \( n + 1 \) points of the form \( x + ihq \) where \( x \) is the base point, \( q \) is the unit directional vector, \( i \) is an integer, \( h = (\epsilon_m)^{\frac{1}{n+2}} \) is the discretization step and \( \epsilon_m \) is the machine precision.

3.1. AB-neuron model. A good example to compare the three kinds of differentiation is the AB-neuron model. The equations and fixed parameter values of this model can be found in [9]. It is a highly complicated nonlinear 6-dimensional system where \( g_A \) and \( g_{K_C} \) are the parameters of interest to us. This model contains several GH-points and two HH-points. The investigated bifurcation points are shown in Figure 1. There is also a BT-point outside the range, with \( g_{K_C} = -0.1240 \) and \( g_A = 263.3610 \).

The first column of Table 2 shows the bifurcation, the others give the elapsed time in four cases. The second column is the case where the bifurcation point and the normal form coefficient are computed symbolically, for the third the bifurcation point is computed with SD and the normal form coefficient is computed with AD, in the fourth and fifth column the bifurcation point is computed with FD, with the normal form coefficient computed with AD in the fourth column and with FD in the last column.

Remarkable is the fact that in all cases the elapsed time needed for AD is much higher than for SD. The elapsed times for SD don’t include the generation of the symbolic derivatives by MATLAB, but are the precise computation times for the specific normal form coefficients. Especially the computation of the HH-points requires much more calculation time, the elapsed time needed for AD is 600 to 900 times that for SD. In the AD case the computation takes about 4 minutes, but be aware that the normal form coefficient of a HH-point consists of 5 coefficients. For the GH-points, SD is about 300 times faster than AD. The elapsed time for a H-point is with SD 400 to 500 times faster than with AD, though the normal form coefficient of a H-point doesn’t require complex calculations. In the other cases the symbolic derivatives also give faster results, but the ratios are not that impressive.
In fact, the elapsed times when using AD are in all cases still acceptable (less or around one second), except in the GH- and HH-cases.

The above results are of course very dependent on the processor speed and computing power. We guess that the high time complexity in the AD case is due to the overhead caused by the implementation of Taylor series as an object type in MATLAB rather than to the number of floating point computations.
Table 3. Column 2: normal form coefficient, Columns 3, 4 and 5: relative error compared to the column 2 case, AB-neuron model.

| bifurcation | bp SD | bp SD | bp FD | bp FD |
|-------------|-------|-------|-------|-------|
|             | nf SD | nf AD | nf AD | nf FD |
| H1          | 3.56477e-4 | 8e-15 | 1e-7  | 8e-5  |
| GH1         | -3.09215e-6 | 3e-14 | 3e-8  | 3e-2  |
| GH2         | 3.12466e-7  | 1e-12 | 6e-8  | 1     |
| GH3         | -2.81207e-5 | 7e-14 | 3e-8  | 3e-3  |
| HH11        | -1     | 0     | 0     | 0     |
| HH12        | 1.96768 | 2e-14 | 3e-7  | 5e-5  |
| HH13        | 4.68922 | 6e-14 | 2e-7  | 1e-4  |
| HH14        | -7.46229e+3 | 4e-13 | 1e-7  | 1e-3  |
| HH15        | -8.71083e+4 | 1e-13 | 5e-7  | 7e-4  |
| GH4         | 4.26750e-3  | 2e-12 | 2e-6  | 2e-2  |
| GH5         | -2.05283   | 2e-13 | 1e-6  | 5e-4  |
| GH6         | -2.70063e-3 | 4e-14 | 5e-8  | 1e-3  |
| HH21        | -1     | 0     | 0     | 0     |
| HH22        | 1.96768 | 5e-13 | 5e-7  | 4e-5  |
| HH23        | 4.68922 | 1e-13 | 2e-7  | 2e-5  |
| HH24        | -7.46229e+3 | 7e-15 | 7e-8  | 1e-3  |
| HH25        | -8.71081e+4 | 1e-14 | 8e-7  | 4e-4  |
| LP1         | -2.25966e-3 | 4e-16 | 1e-6  | 5e-6  |
| LP1         | -1.54358e-4 | 3e-15 | 7e-6  | 7e-6  |
| H2          | 2.48136e-4  | 1e-14 | 8e-8  | 8e-5  |
| CP          | 8.61850e-5  | 7e-14 | 9e-7  | 1e-4  |
| BT1         | -1.23785e-5 | 3e-15 | 1e-7  | 7e-7  |
| BT2         | -7.05817e-5 | 1e-15 | 1e-7  | 4e-7  |

Table 4. Relative distance between the bifurcation points computed by SD and FD, AB-neuron model.

| bifurcation | relative distance | bifurcation | relative distance |
|-------------|-------------------|-------------|-------------------|
| H1          | 1e-10             | GH6         | 3e-8              |
| GH1         | 2e-6              | HH2         | 9e-11             |
| GH2         | 5e-5              | LP1         | 2e-8              |
| GH3         | 3e-5              | LP2         | 6e-7              |
| HH1         | 4e-9              | H2          | 1e-9              |
| GH4         | 2e-7              | CP          | 3e-7              |
| GH5         | 5e-9              | BT          | 7e-9              |

Another remarkable fact is that the elapsed time in SD cases seems to be erratic while in the AD cases the computation times are more consistent.

Table 3 gives an indication of the accuracy of the computed normal form coefficients. We take the normal form coefficient in the second column where both bifurcation point and normal form coefficient are computed symbolically as reference value. The last three columns give the relative error compared to this case. We recall that a BT point consists of 2 normal form coefficients, a ZH point of three and a HH point of 5 coefficients.
Table 5. Elapsed time and relative error in a hybrid method computation, AB-neuron model.

| bifurcation | elapsed time | relative error |
|-------------|--------------|----------------|
| GH1         | 9.506151     | 6e − 11        |
| GH2         | 9.514462     | 2e − 10        |
| GH3         | 9.395758     | 6e − 10        |
| HH1_1       | 98.548896    | 0              |
| HH1_2       | 0            |                |
| HH1_3       | 0            |                |
| HH1_4       | 7e − 4       |                |
| HH1_5       | 2e − 3       |                |
| GH4         | 9.528778     | 4e − 9         |
| GH5         | 9.508569     | 7e − 9         |
| GH6         | 9.483299     | 3e − 9         |
| HH2_1       | 97.599567    | 0              |
| HH2_2       | 0            |                |
| HH2_3       | 0            |                |
| HH2_4       | 4e − 4       |                |
| HH2_5       | 2e − 2       |                |

The normal form coefficients computed by SD and AD with symbolically computed bifurcation point are equal to within machine precision. Indeed, in the calculation of the normal form coefficients, systems are solved with quite high condition number (of the order $1e + 5$). This explains why the results of the second column case are not identical to the results of the third column case.

When the bifurcation point is computed with FD and the normal form coefficient with AD, the results are still acceptable, taking the relative distance between the symbolically and the with FD computed bifurcation points into account, see Table 4. We note that the bifurcation point value computed by SD is not necessarily more accurate than the one obtained by FD. However, when the normal form coefficient is computed with FD (last column in Table 3), very inaccurate results are obtained. In the computation of the second GH-point not even one digit is guaranteed.

Since the computation of the fourth and fifth order derivatives is found to cause a main slow-down in computation time, it is interesting to look at the results of Table 5. Here we have considered the case where the first to third order derivatives are computed with AD, and the fourth and fifth order derivatives by the use of first and second order finite differences on the third order tensor. The bifurcation point is computed with FD. The elapsed time given in the second column in the case of the two HH-points is the computation time for the 5 normal form coefficients together. The elapsed times should be compared with the results from the last but one column of Table 2. We obtain a time improvement of 2.5 times, both for the GH- and the HH-points. However, in comparison with a symbolically computed normal form coefficient, this is still much slower. The last column of Table 5 lists the relative error in comparison with the case where AD is used for all the derivatives. The results for the GH-points are acceptable, but for the HH-points big relative errors are obtained. So the gain of calculation time doesn’t compensate for the loss of accuracy. Therefore, this is not an option to be implemented in MatCont.
3.2. Morris Lecar model. We study now the Morris Lecar model [12], a quite complex system with only two state variables:

\[
\begin{align*}
\dot{V} &= (-g_{Ca}m_\infty(V - V_{Ca}) - g_K w(V - V_K) - g_L(V - V_L) + I_{app})/C, \\
\dot{w} &= \frac{\phi(w_\infty - w)}{\tau}, \\
m_\infty &= 0.5 \left(1 + \tanh \left(\frac{V - v_1}{v_2}\right)\right), \\
w_\infty &= 0.5 \left(1 + \tanh \left(\frac{V - v_3}{v_4}\right)\right), \\
\tau &= \frac{1}{\cosh \left(\frac{V - v_3}{2v_4}\right)}.
\end{align*}
\]

The investigated bifurcations are presented in Figure 2. The fixed parameter-values are \(C = 5, V_K = -80, g_K = 8, V_{Ca} = 120, g_{Ca} = 4, V_L = -60, g_L = 2, v_1 = -1.2, v_2 = 18, v_4 = 17.4, \phi = 1/15.\)
Table 7. Column 2: normal form coefficient, Columns 3, 4 and 5: relative error compared to the column 2 case, Morris Lecar model.

| Bifurcation | BP SD | NF SD | BP SD | NF AD | BP FD | NF AD | BP FD | NF FD |
|-------------|-------|-------|-------|-------|-------|-------|-------|-------|
| H           | 2.37370e-3 | 8e-16 | 5e-9  | 1e-3  |       |       |       |       |
| GH1         | -5.34108e-6 | 1e-14 | 1e-7  | 1     |       |       |       |       |
| GH2         | -7.48731e-5 | 1e-14 | 4e-8  | 2e-1  |       |       |       |       |
| BT1         | -4.62517e-3 | 2e-16 | 1e-9  | 7e-6  |       |       |       |       |
| BT12        | -9.16274e-2 | 0     | 1e-10 | 8e-6  |       |       |       |       |
| BT21        | 5.14675e-3  | 0     | 4e-10 | 4e-6  |       |       |       |       |
| BT22        | 7.30086e-2  | 5e-16 | 3e-10 | 1e-5  |       |       |       |       |
| BT31        | 1.83031e-3  | 2e-15 | 3e-10 | 1e-5  |       |       |       |       |
| BT32        | 5.00787e-2  | 1e-16 | 2e-12 | 6e-6  |       |       |       |       |
| BT41        | -8.26671e-3 | 3e-16 | 6e-10 | 1e-6  |       |       |       |       |
| BT42        | -1.15036e-1 | 2e-16 | 5e-10 | 4e-6  |       |       |       |       |

Table 6 lists the elapsed time in the four interesting cases. Again, the elapsed time needed for the computation of the normal form coefficients with SD is much shorter than with AD. For the GH-points, SD is 200 to 1700 times faster than AD. Although only two multilinear forms of order 2 are required for the BT-points, SD is 4 to 70 times faster than AD. We again note the more consistent elapsed times for SD.

From Table 7 we can derive analogous conclusions concerning the accuracy as for the AB-neuron model. Despite the high condition number of the linear systems to be solved (again order $1e+5$), the results of the normal form coefficients with symbolically computed bifurcation point and normal form coefficient computed either with SD or AD are very similar. Moreover, the multilinear forms are identical when using symbolic or automatic derivatives. The obtained relative errors of the last but one column are acceptable, taking the relative distances between the bifurcation points into account, which are of the same order as in the AB-neuron model. We obtain inaccurate results when using finite differences.

3.3. The Brusselator. Our last example is the 1-dimensional Brusselator:

$$\begin{align*}
\dot{X} &= \frac{D}{L} \frac{\partial^2 X}{\partial z^2} - ((B + 1)X - X^2Y - A), \\
\dot{Y} &= \frac{D}{L} \frac{\partial^2 Y}{\partial z^2} - (X^2Y - BX).
\end{align*}$$

We discretize this system using an equidistant mesh of $n$ intermediate points in the interval $[0, 1]$, with the length of the mesh intervals equal to $h = \frac{1}{n+1}$ and with function values $X_1, Y_1, \ldots, X_n, Y_n$. At the beginning point and end point we take
$X$ equal to $A$ and $Y$ equal to $B/A$. The discretization is then given by

$$
\dot{X}_1 = \frac{D_X}{L^2} \frac{X_1 - 2X_1 + X_2}{h^2} - ((B + 1)X_1 - X_2^2Y_1 - A),
$$

$$
\dot{Y}_1 = \frac{D_Y}{L^2} \frac{B/A - 2Y_1 + Y_2}{h^2} - (X_1^2Y_1 - BX_1),
$$

$$
\vdots
$$

$$
\dot{X}_i = \frac{D_X}{L^2} \frac{X_{i-1} - 2X_i + X_{i+1}}{h^2} - ((B + 1)X_i - X_i^2Y_i - A),
$$

$$
\dot{Y}_i = \frac{D_Y}{L^2} \frac{Y_{i-1} - 2Y_i + Y_{i+1}}{h^2} - (X_i^2Y_i - BX_i),
$$

$$
\vdots
$$

$$
\dot{X}_n = \frac{D_X}{L^2} \frac{X_{n-1} - 2X_n + X_n}{h^2} - ((B + 1)X_n - X_n^2Y_n - A),
$$

$$
\dot{Y}_n = \frac{D_Y}{L^2} \frac{Y_{n-1} - 2Y_n + B/A}{h^2} - (X_n^2Y_n - BX_n).
$$

The point $(X_i, Y_i) = (A, B/A)$, $i = 1, \ldots, n$, is an equilibrium independently of the value of $L$. The parameter values are $D_X = 10$, $D_Y = 1$, $A = 2$, $B = 6$ and $L = 1$. We start the continuation from this equilibrium, with $L$ as free system parameter. The original system (2) has a Hopf point for $L = 10.4194808$ ($= \pi \sqrt{\frac{D_X + D_Y}{A^2 + B/A}}$).

We detect the Hopf point of the discretizations in (3) for several values of $n$. But for $n = 25$, MatCont is not able to compute the normal form coefficient by use of symbolic derivatives, an “Out of memory error” is given. With AD, there are no problems. So AD then forms the only option for the computation of the normal form coefficient of the Hopf point (FD is inaccurate). For $n = 20$, MatCont is able to compute the symbolic derivatives in the Symbolic Toolbox up to order three, but not up to order five, although the fourth and fifth derivatives are identically zero. So in this case restrictions of the software force us to use automatic differentiation. As an alternative one might consider to introduce sparse symbolic derivatives in MatCont (but we note that MATLAB does not support sparse multidimensional arrays).

4. Conclusion. An accurate computation of the normal form coefficients of bifurcations of ODEs is required since these coefficients are used in the detection of higher codimension bifurcations, in switching to branches of lower codimension objects etc. As supported by the examples, SD and AD are equally accurate, while FD give often inaccurate results. Therefore, when the Symbolic Toolbox is not available or, when this toolbox is available but MatCont is not capable to store the symbolic derivatives, automatic differentiation is preferable to finite differences. However, the computation time of the normal form coefficients of GH- and HH-points is often prohibitive. Therefore, we propose that in these two cases the user is given the option to choose whether the normal form coefficient is computed or not. The normal form coefficients of the other equilibrium bifurcations should always be computed and AD should be used if SD is not available.

REFERENCES

[1] C.H. Bischof, H.M. Bücker, B. Lang, A. Rasch and A. Vehreschild, Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs, in Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation, Los Alamitos, (2002), 65-72.
[2] C. Bischof, B. Lang and A. Vehreschild, *Automatic differentiation for MATLAB programs*, in Proceedings of Applied Mathematics and Mechanics, John Wiley, (2003), 50-53.
[3] T.F. Coleman and A. Verma, “ADMAT: An automatic differentiation toolbox for MATLAB,” Tech. rep., Cornell University, 1998.
[4] T.F. Coleman and A. Verma, *The efficient computation of sparse jacobian matrices using automatic differentiation*, SIAM J. Sci. Comput., 19 (1998), 1210–1233.
[5] T.F. Coleman and A. Verma, *ADMIT-1: Automatic differentiation and MATLAB interface toolbox*, ACM Trans. Math. Software, 26 (2000), 150–175.
[6] A. Dhooge, W. Govaerts and Yu.A. Kuznetsov, *MATCONT: A MATLAB package for numerical bifurcation analysis of ODEs*, ACM Trans. Math. Software, 29 (2003), 141–164. Available via http://sourceforge.net/projects/matcont/.
[7] A. Dhooge, W. Govaerts, Yu.A. Kuznetsov, H.G.E. Meijer and B. Sautois, *New features of the software MatCont for bifurcation analysis of dynamical systems*, Math. Comput. Model. Dyn. Syst., 14 (2008), 147–175.
[8] S.A. Forth, “User Guide for MAD - a Matlab Automatic Differentiation Toolbox,” Applied Mathematics and Operational Research Report, Cranfield University in Swindon, UK, 2001.
[9] W. Govaerts and A. Dhooge, *A two-parameter numerical study of the neuron model of Plant*, in Proceedings of the 3rd Scientific Colloquium, Prague, (2001), 9–21.
[10] A. Griewank, “Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation,” SIAM, Philadelphia, 2000.
[11] A. Griewank and A. Walther, “Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation,” 2nd edition, SIAM, Philadelphia, 2008.
[12] C. Morris and H. Lecar, *Voltage oscillations in the barnacle giant muscle-fiber*, Biophys. J., 35 (1981), 193–213.
[13] J.D. Pryce, R. Khoshsiar Ghaziani, V. De Witte and W. Govaerts, *Computation of normal form coefficients of cycle bifurcations of maps by algorithmic differentiation*, Math. Comput. Simulation, 81 (2010), 109–119.
[14] L.C. Rich and D.R. Hill, *Automatic differentiation in MATLAB*, Appl. Numer. Math., 9 (1992), 33–43.
[15] A. Verma, “Structured automatic differentiation,” Ph.D. thesis, Cornell University in Ithaca, NY, 1998.

Received July 2010; revised August 2010.

E-mail address: Virginie.DeWitte@UGent.be
E-mail address: Willy.Govaerts@UGent.be