Mathematical Biology and Bioinformatics
2017. V. 12. № 2. P. 496–520. doi: /10.17537/2017.12.469

==========**MATHEMATICAL MODELING** ==========

UDC: 519.6

Numerical bifurcation analysis of mathematical models with time delays with the package DDE-BIFTOOL

©2017 Luzyanina T.[1], Sieber J.[2], Engelborghs K.[3], Samaey G.[4], Roose D.[4]

1 *Institute of Mathematical Problems of Biology - the branch of Keldysh Institute of Applied Mathematics, 142290 Pushchino, Russia*

[2] *Department of Mathematics, University of Exeter, Exeter EX4 4QF, UK*

[3] *Materialise NV, Technologielaan 15, 3001 Leuven, Belgium*

[4] *Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200 A, B-3001 Heverlee-Leuven, Belgium*

Abstract. Mathematical modelling with delay differential equations (DDEs) is widely used for analysis and predictions in various areas of the life sciences, e.g., population dynamics, epidemiology, immunology, physiology, neural networks. The time delays in these models take into account a dependence of the present state of the modelled system on its past history. The delay can be related to the duration of certain hidden processes like the stages of the life cycle, the time between infection of a cell and the production of new viruses, the duration of the infectious period, the immune period and so on. Due to an infinite-dimensional nature of DDEs, analytical studies of the corresponding mathematical models can only give limited results. Therefore, a numerical analysis is the major way to achieve both a qualitative and quantitative understanding of the model dynamics. A bifurcation analysis of a dynamical system is used to understand how solutions and their stability change as the parameters in the system vary. The package DDE-BIFTOOL is the first general-purpose package for bifurcation analysis of DDEs. This package can be used to compute and analyze the local stability of steady-state (equilibria) and periodic solutions of a given system as well as to study the dependence of these solutions on system parameters via continuation. Further one can compute and continue several local and global bifurcations: fold and Hopf bifurcations of steady states; folds, period doublings and torus bifurcations of periodic orbits; and connecting orbits between equilibria. In this paper we describe the structure of DDE-BIFTOOL, numerical methods implemented in the package and we illustrate the use of the package using a certain DDE system.

Key words: *nonlinear dynamics, delay differential equations, stability analysis, periodic solutions, collocation methods, numerical bifurcation analysis, state-dependent delay.*

1. INTRODUCTION

This paper takes most of its material from a sequence of documents, written by the authors, that have evolved over time in the form of handbooks for the software DDE-BIFTOOL, see [19] and http://arxiv.org/abs/1406.7144 for most up-to-date version of DDE-BIFTOOL and the references there in. The material in sections 2 to 7 has been made available previously by the authors as documentation for DDE-BIFTOOL version 3.1.1 on http://arxiv.org/abs/1406.7144v4. The example in section 8 has been extracted from an earlier version of the documentation [19]. The code reproducing the results of section 8 is freely

available at http://ddebiftool.sourceforge.net/demos.

2. DELAY DIFFERENTIAL EQUATIONS

This section introduces the mathematical notation that we refer to in this paper to describe the problems solved by DDE-BIFTOOL.

2.1. Equations with constant delays

Consider the system of delay differential equations with constant delays (DDEs),

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = f(x(t), x(t-\tau_1), \ldots, x(t-\tau_m), \eta), \tag{1}$$

where $x(t) \in \mathbb{R}^n$, $f : \mathbb{R}^{n(m+1)} \times \mathbb{R}^p \to \mathbb{R}^n$ is a nonlinear smooth function depending on a number of parameters $\eta \in \mathbb{R}^p$, and delays $\tau_i > 0$, $i = 1, \ldots, m$. Call τ the maximal delay,

$$\tau = \max_{i=1,\ldots,m} \tau_i.$$

The linearization of (1) around a solution $x^*(t)$ is the *variational equation*, given by,

$$\frac{\mathrm{d}}{\mathrm{d}t}y(t) = A_0(t)y(t) + \sum_{i=1}^{m} A_i(t)y(t-\tau_i), \tag{2}$$

where, using $f \equiv f(x^0, x^1, \ldots, x^m, \eta)$,

$$A_i(t) = \frac{\partial f}{\partial x^i}(x^*(t), x^*(t-\tau_1), \ldots, x^*(t-\tau_m), \eta),\ i = 0, \ldots, m. \tag{3}$$

2.1.1. Steady states

If $x^*(t)$ corresponds to a steady state solution,

$$x^*(t) \equiv x^* \in \mathbb{R}^n, \text{ with } f(x^*, x^*, \ldots, x^*, \eta) = 0,$$

then the matrices $A_i(t)$ are constant, $A_i(t) \equiv A_i$, and the corresponding variational equation (2) leads to a *characteristic equation*. Define the $n \times n$-dimensional matrix Δ as

$$\Delta(\lambda) = \lambda I - A_0 - \sum_{i=1}^{m} A_i e^{-\lambda\tau_i}. \tag{4}$$

Then the characteristic equation reads,

$$\det(\Delta(\lambda)) = 0. \tag{5}$$

Equation (5) has an infinite number of roots $\lambda \in \mathbb{C}$ which determine the stability of the steady state solution x^*. The steady state solution is (asymptotically) stable provided all roots of the characteristic equation (5) have negative real part; it is unstable if there exists a root with positive real part. It is known that the number of roots in any right half plane $\mathrm{Re}(\lambda) > \gamma$, $\gamma \in \mathbb{R}$ is finite, hence, the stability is always determined by a finite number of roots.

Bifurcations occur whenever roots move through the imaginary axis as one or more parameters are changed. Generically a fold bifurcation (or turning point) occurs when the root is real (that is, equal to zero) and a Hopf bifurcation occurs when a pair of complex conjugate roots crosses the imaginary axis.

2.1.2. Periodic orbits

A periodic solution $x^*(t)$ is a solution which repeats itself after a finite time, that is,

$$x^*(t+T) = x^*(t), \text{ for all } t.$$

Here $T > 0$ is the period. The stability around the periodic solution is determined by the time integration operator $S(T,0)$ which integrates the variational equation (2) around $x^*(t)$ from time $t = 0$ over the period. This operator is called the *monodromy operator* and its (infinite number of) eigenvalues, which are independent of the starting moment $t = 0$, are called the *Floquet multipliers*. Furthermore, if $S(T,0)^k$ is compact for $k > \tau/T$. Thus, there are at most finitely many Floquet multipliers outside of any ball around the origin of the complex plane.

For autonomous systems there is always a *trivial* Floquet multiplier at unity, corresponding to a perturbation along the time derivative of the periodic solution. The periodic solution is exponentially stable provided all multipliers (except the trivial one) have modulus smaller than unity, it is exponenially unstable if there exists a multiplier with modulus larger than unity.

2.1.3. Connecting orbits

We call a solution $x^*(t)$ of (1) at $\eta = \eta^*$ a *connecting orbit* if the limits

$$\lim_{t\to-\infty} x^*(t) = x^-, \qquad \lim_{t\to+\infty} x^*(t) = x^+, \tag{6}$$

exist. For continuous f, x^- and x^+ are steady state solutions. If $x^- = x^+$, the orbit is called homoclinic, otherwise it is heteroclinic.

2.2. Equations with state-dependent delays

Consider the system of delay differential equations with state-dependent delays (sd-DDEs),

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}x(t) = f(x_0, x_1, \ldots, x_m, \eta), \\ \qquad x_j = x(t - \tau_j(x_0, \ldots, x_{j-1}, \eta) \quad (\tau_0 = 0, j = 1, \ldots, m), \end{cases} \tag{7}$$

where $x(t) \in \mathbb{R}^n$, and

$$\begin{aligned} f &: \mathbb{R}^{n(m+1)} \times \mathbb{R}^p \to \mathbb{R}^n \\ \tau_j &: \mathbb{R}^{nj} \times \mathbb{R}^p \to [0, \infty) \end{aligned}$$

are smooth functions depending of their arguments. The right-hand side f depends on $m+1$ states $x_j = x(t-\tau_j) \in \mathbb{R}^n$ ($j = 0, \ldots, m$) and p parameters $\eta \in \mathbb{R}^p$. The jth delay function τ_j depends on all previously defined $j-1$ states $x(t-\tau_i) \in \mathbb{R}^n$ ($i = 0, \ldots, j-1$) and p parameters $\eta \in \mathbb{R}^p$. This definition permits the user to formulate sd-DDEs with arbitrary levels of nesting in their function arguments.

The linearization around a solution $(x^*(t), \eta^*)$ of (7) (the *variational equation*) with respect to x is given by (see [29], we are using the notation $x_0^* = x^*(t)$, $\tau_j^*(t) = \tau_j(x_0^*, \ldots, x_{j-1}^*, \eta^*)$

and $x_j^* = x^*(t - \tau_j^*(t))$ for $j \geq 1$)

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} y(t) &= \sum_{j=0}^{m} A_j(t) Y_k \\ Y_0 &= y(t) \\ Y_j &= y(t - \tau_j^*(t)) - (x^*)'(t - \tau_j(t)) \sum_{k=0}^{j-1} B_{k,j}(t) Y_k, \quad (j = 1 \ldots m) \end{aligned} \tag{8}$$

where $(x^*)'(t) = \mathrm{d}x^*(t)/\mathrm{d}t$, and

$$\begin{aligned} A_j(t) &= \frac{\partial f}{\partial x^j}(x_0^*, x_1^*, \ldots, x_m^*, \eta) \in \mathbb{R}^{n \times n}, \quad (j = 0, \ldots, m), \\ B_{j,k}(t) &= \frac{\partial \tau_j}{\partial x^k}(x_0^*, x_1^*, \ldots, x_{j-1}^*, \eta^*) \in \mathbb{R}^{1 \times n}, \quad (k = 0, \ldots, j-1, j = 1, \ldots, m). \end{aligned} \tag{9}$$

If $(x^*(t), \tilde{\tau}^*(t))$ corresponds to a steady state solution, then $x^*(t) = x_0^* = \ldots = x_m^* \equiv x^* \in \mathbb{R}^n$, and $\tau_j^*(t) \equiv \tau_j(x^*, \ldots, x^*, \eta^*)$ for all $j \geq 1$, with

$$f(x^*, x^*, \ldots, x^*, \eta^*) = 0$$

then the matrices $A_i(t)$ are constant, $A_i(t) \equiv A_i$, and the vectors $B_{i,j}(t)$ consist of zero elements only. In this case, the corresponding variational equation (8) is a constant delay differential equation and it leads to the characteristic equation (5), i.e. a characteristic equation with constant delays. Hence the stability analysis of a steady state solution of (7) is similar to the stability analysis of (1).

Note that the right-hand side f, when considered as a functional mapping a history segment into $\mathbb{R}^n$ is not locally Lipschitz continuous. This creates technical difficulties when considering an sd-DDE of type (7) as an infinite-dimensional system, because the solution does not depend smoothly on the initial condition (see Hartung et al [29] for a detailed review). However, periodic boundary-value problems for (7) can be reduced to finite-dimensional systems of algebraic equations that are as smooth as the coefficient functions f and τ_j [45]. This implies that all periodic orbits and their bifurcations and stability as computed by DDE-BIFTOOL behave as expected. In particular, branching off at Hopf bifurcations and period doubling works in the same way as for constant delays (the proof for the Hopf bifurcation in sd-DDEs is also given in [45]).

Moreover, Mallet-Paret and Nussbaum [38] proved that the stability of the linear variational equation (8) indeed reflects the local stability of the solution $(x^*(t), \tilde{\tau}^*(t))$ of (7). For details on the relevant theory and numerical bifurcation analysis of sd-DDEs see [36, 29] and the references therein.

3. CAPABILITIES OF DDE-BIFTOOL

Related software. A large number of packages exist for numerical continuation and bifurcation analysis of systems of ordinary differential equations (ODEs). Currently maintained packages are

AUTO	url: http://sourceforge.net/projects/auto-07p	using FORTRAN or C [12, 11],
MatCont	url: http://sourceforge.net/projects/matcont/	for Matlab [9, 24],
Coco	url: http://sourceforge.net/projects/cocotools	for Matlab [8].

For delay differential equations the package

knut	url: http://gitorious.org/knut	using C++

(formerly PDDECONT) is available as a stand-alone package (written in C++, but with a user interface requiring no programming). This package was developed in parallel with DDE-BIFTOOL but independently by R. Szalai [46, 40].

For simulation (time integration) of delay differential equations the reader is, e.g., referred to the packages ARCHI, DKLAG6, XPPAUT, DDVERK, RADAR and dde23, see [39, 7, 23, 22, 43, 26]. Of these, only XPPAUT has a graphical interface (and allows limited stability analysis of steady state solutions of DDEs along the lines of [37]). TRACE-DDE is a Matlab tool (with graphical interface) for linear stability analysis of linear constant-coefficient DDEs [5].

DDE-BIFTOOL. The package DDE-BIFTOOL is freely available for scientific (non-commercial) use (see https://sourceforge.net/projects/ddebiftool/ to download the package). It was started by K. Engelborghs as part of his PhD at the Computer Science Department of the K.U.Leuven under supervision of Prof. D. Roose. T. Luzyanina extended the package to delay differential equations with state-dependent delays. Computation of heteroclinic and homoclinic solutions of DDEs was implemented by G. Samaey. J. Sieber implemented continuation of local bifurcations of periodic orbits for DDEs with constant and state-dependent delays and is the current maintainer. S. Janssens, B. Wage, M. Bosschaert and Yu. Kuznetsov contributed the normal form analysis capabilities for equilibria of DDEs.

DDE-BIFTOOL consists of a set of routines running in Matlab [31] or octave, both widely used environments for scientific computing. The aim of the package is to provide a tool for numerical bifurcation analysis of steady state solutions and periodic solutions of DDEs with constant delays or state-dependent delays (sd-DDEs). It also allows users to compute homoclinic and heteroclinic orbits in DDEs (with constant delays).

DDE-BIFTOOL can perform the following computations:

- continuation of steady state solutions (typically in a single parameter);
- approximation of the rightmost, stability-determining roots of the characteristic equation which can further be corrected using a Newton iteration;
- continuation of steady state folds and Hopf bifurcations (typically in two system parameters);
- continuation of periodic orbits using orthogonal collocation with adaptive mesh selection (starting from a previously computed Hopf point or an initial guess of a periodic solution profile);
- approximation of the largest stability-determining Floquet multipliers of periodic orbits;
- branching onto the secondary branch of periodic solutions at a period doubling bifurcation or a branch point;
- continuation of folds, period doublings and torus bifurcations (typically in two system parameters);
- computation of normal form coefficients for Hopf bifurcations and codimension-two bifurcations along Hopf bifurcation curves (typically in two system parameters);
- continuation of connecting orbits (using the appropriate number of parameters);
- continuation of relative equilibria and relative periodic orbits and their local bifurcations for systems with constant delays and rotational symmetry (saddle-node bifurcation, Hopf bifurcation, period-doubling, and torus bifurcation).

All computations can be performed for problems with an arbitrary number of discrete delays. These delays can be either parameters or functions of the state. (The only exception are computations of connecting orbits, which support only problems with delays as parameters at the moment.)

A practical difference to AUTO or MatCont is that the package does not detect bifurcations automatically because the computation of eigenvalues or Floquet multipliers may require more computational effort than the computation of the equilibria or periodic orbits (for example, if the system dimension is small but one delay is large). Instead the evolution of the eigenvalues can be computed along solution branches in a separate step if required. This allows the user to detect and identify bifurcations.

The current version of the package DDE-BIFTOOL is 3.1.1. Earlier versions of DDE-BIFTOOL continue to be available at the web addresses

versions ≥ 3.0	http://arxiv.org/abs/1406.7144
versions ≤ 2.03	http://twr.cs.kuleuven.be/research/software/delay/ddebiftool.shtml

The manual for the package DDE-BIFTOOL with the corresponding license can be downloaded at https://sourceforge.net/p/ddebiftool/code/HEAD/tree/trunk/dde_biftool/manual/. Scientific publications for which the package DDE-BIFTOOL has been used shall mention usage of the package DDE-BIFTOOL, and shall cite [18] and the current manual to ensure proper attribution and reproducibility.

In the rest of this paper we assume the reader is familiar with the notion of DDEs and with the basic concepts of bifurcation analysis for ODEs. The theory on DDEs and a large number of examples are described in several books. Most notably the early [4, 13, 14, 27, 34] and the more recent [2, 32, 28, 10, 33]. Several excellent books contain introductions to dynamical systems and bifurcation theory of ODEs, see, e.g., [1, 6, 25, 35, 42].

The tutorial demos neuron and sd_demo, providing a step-by-step walk-through for the typical working mode with DDE-BIFTOOL are included as separate html files, published directly from the comments in the demo code. See http://ddebiftool.sourceforge.net/demos/ for links to all demos, many of which are extensively commented.

The DDE-BIFTOOL package can also be used in octave, see the package manual. For an up-to-date list of known differences in syntax and semantics between Matlab and octave see http://www.gnu.org/software/octave.

4. STRUCTURE OF DDE-BIFTOOL

The structure of the package is depicted in figure 1. It consists of four layers.

Layer 0 contains the system definition and consists of routines which allow to evaluate the right hand side f and its derivatives, state-dependent delays and their derivatives and to set or get the parameters and the constant delays. It should be provided by the user and is explained in more detail in section 5. All user-provided functions are collected in a single structure (called `funcs` in this text), and are passed on by the user as arguments to layer-3 or layer-2 functions.

Layer 1 forms the numerical core of the package and is (normally) not directly accessed by the user. The numerical methods used are explained briefly in section 7, more details can be found in the papers [37, 20, 17, 16, 21, 36, 41] and in [15]. Its functionality is hidden by and used through layers 2 and 3.

Layer 2 contains routines to manipulate individual points. of the following five types. It can be a steady state point (abbreviated `'stst'`), steady state Hopf (abbreviated `'hopf'`) or fold (abbreviated `'fold'`) bifurcation point, a periodic solution point (abbreviated `'psol'`) or a connecting orbit point (abbreviated `'hcli'`). Furthermore, a point can contain additional

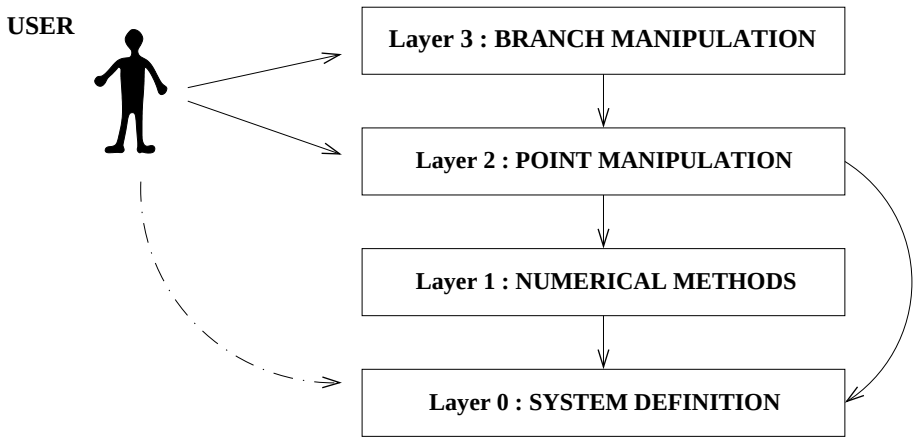

Fig. 1. The structure of DDE-BIFTOOL. Arrows indicate the calling (−) or writing (·−) of routines in a certain layer.

information concerning its stability. Routines are provided to compute individual points, to compute and plot their stability and to convert points from one type to another.

Layer 3 contains routines to manipulate branches. A branch is structure containing an array of (at least two) points, three sets of method parameters and specifications concerning the free parameters. The `'point'` field of a branch contains an array of points of the same type ordered along the branch. The `'method'` field contains parameters of the computation of individual points, the continuation strategy and the computation of stability. The `'parameter'` field contains specification of the free parameters (which are allowed to vary along the branch), parameter bounds and maximal step sizes. Routines are provided to extend a given branch (that is, to compute extra points using continuation), to (re)compute stability along the branch and to visualize the branch and/or its stability.

Layers 2 and 3 require specific data structures, explained in the manual on DDE-BIFTOOL, to represent points, stability information, branches, to pass method parameters and to specify plotting information. Usage of these layers is demonstrated through a step-by-step analysis of the demo systems neuron, sd_demo and hom_demo (see http://ddebiftool.sourceforge.net/demos/). Descriptions of input/output parameters and functionality of all routines in layers 2 and 3 are also given in the manual.

5. SYSTEM DEFINITION

Note that in DDE-BIFTOOL all user-provided functions can be arbitrary function handles, collected into a structure using the function `set_funcs`, explained in section 5.3. The only typical mandatory functions for the user to provide are the right-hand side (`'sys_rhs'`) and the function returning the delay indices (`'sys_tau'`). The names of the user functions can be arbitrary, and user functions can be anonymous. See the tutorials in http://ddebiftool.sourceforge.net/demos/ for examples of usage, and the function description in section 5.3 for details.

5.1. Equations with constant delays

As an illustrative example we will use the following system of DDEs, taken from [44] and called ”neuron” in demo examples

$$\begin{cases} \dot{x_1}(t) = -\kappa x_1(t) + \beta \tanh(x_1(t-\tau_s)) + a_{12} \tanh(x_2(t-\tau_2)) \\ \dot{x_2}(t) = -\kappa x_2(t) + \beta \tanh(x_2(t-\tau_s)) + a_{21} \tanh(x_1(t-\tau_1)). \end{cases} \tag{10}$$

This system models two coupled neurons with time delayed connections. It has two components (x_1 and x_2), three delays (τ_1, τ_2 and τ_s), and four parameters (κ, β, a_{12} and a_{21}). The demo neuron walks through the bifurcation analysis of system (10) step by step to demonstrate the working pattern for DDE-BIFTOOL.

To define a system, the user should provide the following Matlab functions, given in the following paragraphs for system (10).

5.1.1. Right-hand side — sys_rhs

The right-hand side is a function of two arguments. For our example (10), this would have the form (giving the right-hand side the name neuron_sys_rhs)

```
neuron_sys_rhs=@(xx,par)[...
  -par(1)*xx(1,1)+par(2)*tanh(xx(1,4))+par(3)*tanh(xx(2,3));...
  -par(1)*xx(2,1)+par(2)*tanh(xx(2,4))+par(4)*tanh(xx(1,2))];
%par=[\kappa,\beta, a_{12}, a_{21},\tau_1,\tau_2, \tau_s]
```

Listing 1. Definition for right-hand side of (10) as a variable.

Meaning of the arguments of the right-hand side function:

- xx $\in \mathbb{R}^{n\times(m+1)}$ contains the state variable(s) at the present and in the past,
- par $\in \mathbb{R}^{1\times p}$ contains the parameters, par $= \eta$.

The delays τ_i ($i = 1\ldots,m$) are considered to be part of the parameters ($\tau_i = \eta_{j(i)}$, $i = 1,\ldots,m$). This is natural since the stability of steady solutions and the position and stability of periodic solutions depend on the values of the delays. Furthermore delays can occur both as a 'physical' parameter and as delay, as in $\dot{x} = \tau x(t-\tau)$. From these inputs the right hand side f is evaluated at time t. Notice that the parameters have a specific order in par indicated in the comment line.

An alternative (vectorized) form would be

```
neuron_sys_rhs=@(xx,p)[...
 -p(1)*xx(1,1,:)+p(2)*tanh(xx(1,4,:))+p(3)*tanh(xx(2,3,:));...
 -p(1)*xx(2,1,:)+p(2)*tanh(xx(2,4,:))+p(4)*tanh(xx(1,2,:))];
```

Listing 2. Alternative definition of the right-hand side of (10), vectorized for speed-up of periodic orbit computations.

Note the additional colon in argument xx and compare to Listing 1. The form shown in Listing 2 can be called in many points along a mesh simultaneously, speeding up the computations during analysis of periodic orbits.

5.1.2. Delays – sys_tau

For constant delays another function is required which returns the *position* of the delays in the parameter list. For our example, this is

```
neuron_tau=@()[5 6 7];
```

This function has no arguments for constant delays, and returns a row vector of indices into the parameter vector.

5.1.3. Jacobians of right-hand side — `sys_deri` (optional, but recommended)

Several derivatives of the right hand side function f need to be evaluated during bifurcation analysis. By default, DDE-BIFTOOL uses a finite-difference approximation, implemented in `df_deriv.m`. For speed-up or in case of convergence difficulties the user may provide the Jacobians of the right-hand side analytically as a separate function. Its header is of the format

```
function J=sys_deri(xx,par,nx,np,v)
```

Arguments:

- `xx` $\in \mathbb{R}^{n\times(m+1)}$ contains the state variable(s) at the present and in the past (as for the right-hand side);
- `par` $\in \mathbb{R}^{1\times p}$ contains the parameters, `par` $= \eta$ (as for the right-hand side);
- `nx` (empty, one integer or two integers) index (indices) of `xx` with respect to which the right-hand side is to be differentiated
- `np` (empty or integer) whether right-hand side is to be differentiated with respect to parameters
- `v` (empty or $\mathbb{C}^n$) for mixed derivatives with respect to `xx`, only the product of the mixed derivative with `v` is needed.

The result `J` is a matrix of partial derivatives of f which depends on the type of derivative requested via `nx` and `np` multiplied with `v` (when nonempty), see table 1.

`length(nx)`	`length(np)`	`v`	`J`
1	0	empty	$\frac{\partial f}{\partial x^{\texttt{nx(1)}}} = A_{\texttt{nx(1)}} \in \mathbb{R}^{n\times n}$
0	1	empty	$\frac{\partial f}{\partial \eta_{\texttt{np(1)}}} \in \mathbb{R}^{n\times 1}$
1	1	empty	$\frac{\partial^2 f}{\partial x^{\texttt{nx(1)}}\partial \eta_{\texttt{np(1)}}} \in \mathbb{R}^{n\times n}$
2	0	$\in \mathbb{C}^{n\times 1}$	$\frac{\partial}{\partial x^{\texttt{nx(2)}}}\left(A_{\texttt{nx(1)}} v\right) \in \mathbb{C}^{n\times n}$

Table 1. Results of the function `sys_deri` depending on its input parameters `nx`, `np` and `v` using $f \equiv f(x^0, x^1, \ldots, x^m, \eta)$.

`J` is defined as follows. Initialize `J` with f. If `nx` is nonempty take the derivative of `J` with respect to those arguments listed in `nx`'s entries. Each entry of `nx` is a number between 0 and m based on $f \equiv f(x^0, x^1, \ldots, x^m, \eta)$. E.g., if `nx` has only one element take the derivative with respect to $x^{\texttt{nx(1)}}$. If it has two elements, take, of the result, the derivative with respect to $x^{\texttt{nx(2)}}$ and so on. Similarly, if `np` is nonempty take, of the resulting `J`, the derivative with respect to $\eta_{\texttt{np(i)}}$ where i ranges over all the elements of `np`, $1 \le i \le p$. Finally, if v is not an empty vector multiply the result with v. The latter is used to prevent `J` from being a tensor if two derivatives with respect to state variables are taken (when `nx` contains two elements). Not all possible combinations of these derivatives have to be provided. In the current version, `nx` has at most two elements and `np` at most one. The possibilities are further restricted as listed in table 1.

In the last row of table 1 the elements of `J` are given by,

$$\texttt{J}_{i,j} = \left[\frac{\partial}{\partial x^{\texttt{nx(2)}}} A_{\texttt{nx(1)}} v\right]_{i,j} = \frac{\partial}{\partial x_j^{\texttt{nx(2)}}} \left(\sum_{k=1}^{n} \frac{\partial f_i}{\partial x_k^{\texttt{nx(1)}}} v_k\right),$$

with A_l as defined in (3).

The resulting routine is quite long, even for the small system (10); see http://ddebiftool.sourceforge.net/demos/neuron/html/neuron_sys_deri.html for a printout of the function body. Furthermore, implementing so many derivatives is an activity prone to a number of typing mistakes. Hence a default routine `df_deriv` is available which implements finite difference formulas to approximate the requested derivatives (using several calls to the right-hand side). It is, however, recommended to provide at least the first order derivatives with respect to the state variables using analytical formulas. These derivatives occur in the determining systems for fold and Hopf bifurcations and for connecting orbits, and in the computation of characteristic roots and Floquet multipliers. All other derivatives are only necessary in the Jacobians of the respective Newton procedures and thus influence only the convergence speed.

5.2. Equations with state-dependent delays

DDE-BIFTOOL also permits the delays to depend on parameters and the state. If at least one delay is state-dependent then the format and semantics of the function specifying the delays, `sys_tau`, is different from the format used for constant delays in section 5.1.2 (it now provides the *values* of the delays).

As an illustrative example we will use the following system of DDEs, named `sd_demo` in demo examples,

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}x_1(t) &= \frac{1}{p_1 + x_2(t)}\left(1 - p_2 x_1(t)x_1(t-\tau_3)x_3(t-\tau_3) + p_3 x_1(t-\tau_1)x_2(t-\tau_2)\right),\\
\frac{\mathrm{d}}{\mathrm{d}t}x_2(t) &= \frac{p_4 x_1(t)}{p_1 + x_2(t)} + p_5 \tanh(x_2(t-\tau_5)) - 1,\\
\frac{\mathrm{d}}{\mathrm{d}t}x_3(t) &= p_6(x_2(t) - x_3(t)) - p_7(x_1(t-\tau_6) - x_2(t-\tau_4))e^{-p_8\tau_5},\\
\frac{\mathrm{d}}{\mathrm{d}t}x_4(t) &= x_1(t-\tau_4)e^{-p_1\tau_5} - 0.1,\\
\frac{\mathrm{d}}{\mathrm{d}t}x_5(t) &= 3(x_1(t-\tau_2) - x_5(t)) - p_9,
\end{aligned} \tag{11}$$

where

$$\tau_1, \tau_2 \text{ are constant delays,} \tag{12}$$

$$\tau_3 = 2 + p_5\tau_1 x_2(t)x_2(t-\tau_1), \tag{13}$$

$$\tau_4 = 1 - \frac{1}{1 + x_1(t)x_2(t-\tau_2)}, \tag{14}$$

$$\tau_5 = x_4(t), \tag{15}$$

$$\tau_6 = x_5(t). \tag{16}$$

This system has five components $(x_1, \ldots, x_5)$, six delays $(\tau_1, \ldots, \tau_6)$ and eleven parameters $(p_1, \ldots, p_{11})$, where $p_{10} = \tau_1$ and $p_{11} = \tau_2$.

The Matlab functions provided by the user to define sd-DDE (11) and a step-by-step tutorial

for its analysis are given in demo sd_demo (see http://ddebiftool.sourceforge.net/demos/).

5.3. Collecting user functions into a structure

The user-provided functions are passed on as an additional argument to all routines of DDE-BIFTOOL (similar to standard Matlab routines such as `ode45`). The additional argument is a structure `funcs` containing all the handles to all user-provided functions. In order to create this structure the user is recommended to call the function `set_funcs` at the beginning of the script performing the bifurcation analysis:

```
function funcs=set_funcs(...)
```

Its argument format is in the form of name-value pairs in arbitrary order. For the example (10) of a neuron, discussed in section 5.1 and in demo neuron, the call to `set_funcs` could look as follows:

```
funcs=set_funcs('sys_rhs',neuron_sys_rhs,'sys_tau',@()[5,6,7],...
                                  'sys_deri',@neuron_sys_deri);
```

Note that `neuron_sys_rhs` is a variable (a function handle pointing to an anonymous function defined as in section 5.1.1), and `neuron_sys_deri.m` is the filename in which the function providing the system derivatives are defined (see section 5.1.3). The delay function `'sys_tau'` is directly specified as an anonymous function in the call to `set_funcs` (not needing to be defined in a separate file or as a separate variable). If one does wish to not provide analytical derivatives, one may drop the `'sys_deri'` pair (then a finite-difference approximation, implemented in `df_deriv`, is used):

```
funcs=set_funcs('sys_rhs',neuron_sys_rhs,'sys_tau',@()[5,6,7]);
```

An example for a necessary modification of the right-hand side to permit vectorization is given for the neuron example in Listing 2. The output `funcs` is a structure containing all user-provided functions and defaults for the Jacobians if they are not provided. This output is passed on as first argument to all DDE-BIFTOOL routines during bifurcation analysis.

6. DATA STRUCTURES

To avoid many technical details, in this paper we do not describe the data structures used to define the problem (a system of DDEs or sd-DDEs), and to present individual points (a single steady state, fold, Hopf, periodic and homoclinic/heteroclinic solution), stability information (roots of the characteristic equation or Floquet multipliers), branches of points and plotting information. We also do not describe here point and branch manipulation routines. These are presented in detail in the manual on DDE-BIFTOOL and demo examples (http://ddebiftool.sourceforge.net/demos/).

To have an idea for the reader about such data structures we describe here how method parameters are presented in DDE-BIFTOOL.

To compute a single steady state, fold, Hopf, periodic or connecting orbit solution point, several method parameters have to be passed to the appropriate routines. These parameters are collected into a structure with the fields given in Table 2. For the computation of periodic solutions, additional fields are necessary, marked with and asterisk (*) in Table 2. The meaning of the different fields in Table 2 is explained in the manual.

Parameters controlling the pseudo-arclength continuation (using secant approximations for tangents) are stored in a cetain structure, see the manual. Similarly, for the approximation and correction of roots of the characteristic equation respectively for the computation of the Floquet multipliers method parameters are passed using a structure of a certain form, see the manual.

Table 2. Point method structure: fields and possible values. When different, default values are given in the order 'stst','fold','hopf','psol', 'hcli'. Fields marked with and asterisk (*) are needed and present for points of type 'psol' and 'hcli' only

field	content	default value
'newton_max_iterations'	$\mathbb{N}_0$	5, 5, 5, 5, 10
'newton_nmon_iterations'	$\mathbb{N}$	1
'halting_accuracy'	$\mathbb{R}^+$	1e−10, 1e−9, 1e−9, 1e−8, 1e−8
'minimal_accuracy'	$\mathbb{R}_0^+$	1e−8, 1e−7, 1e−7, 1e−6, 1e−6
'extra_condition'	$\{0,1\}$	0
'print_residual_info'	$\{0,1\}$	0
*'phase_condition'	$\{0,1\}$	1
*'collocation_parameters'	$[0,1]^d$ or empty	empty
*'adapt_mesh_before_correct'	$\mathbb{N}$	0
*'adapt_mesh_after_correct'	$\mathbb{N}$	3

7. NUMERICAL METHODS

This section contains short descriptions of the numerical methods for DDEs and the method parameters used in DDE-BIFTOOL. More details on the methods can be found in the articles [37, 20, 17, 16, 21, 41] or in [15]. For details on applying these methods to bifurcation analysis of sd-DDEs see [36].

7.1. Determining systems

Below we state the determining systems used to compute and continue steady state solutions, steady state fold and Hopf bifurcations, periodic solutions and connecting orbits of systems of delay differential equations.

For each determining system we mention the number of free parameters necessary to obtain (generically) isolated solutions. In the package, the necessary number of free parameters is further raised by the number of steplength conditions plus the number of extra conditions used. This choice ensures the use of square Jacobians during Newton iteration. If, on the other hand, the number of free parameters, steplength conditions and extra conditions are not appropriately matched, Newton iteration solves systems with a non-square Jacobian (for which Matlab uses an over- or under-determined least squares procedure). If possible, it is better to avoid such a situation.

Steady state solutions. A steady state solution $x^* \in \mathbb{R}^n$ is determined from the following n-dimensional determining system with no free parameters.

$$f(x^*, x^*, \ldots, x^*, \eta) = 0. \tag{17}$$

Steady state fold bifurcations. Fold bifurcations, $(x^* \in \mathbb{R}^n, v \in \mathbb{R}^n)$ are determined from the following $2n+1$-dimensional determining system using one free parameter.

$$\begin{aligned} 0 &= f(x^*, x^*, \ldots, x^*, \eta) \\ 0 &= \Delta(x^*, \eta, 0)v \\ 0 &= c^{\mathrm{T}} v - 1 \end{aligned} \tag{18}$$

(see (4) for the definition of the characteristic matrix Δ). Here, $c^{\mathrm{T}} v - 1 = 0$ presents a suitable normalization of v. The vector $c \in \mathbb{R}^n$ is chosen as $c = v^{(0)}/({v^{(0)}}^{\mathrm{T}} v^{(0)})$, where $v^{(0)}$ is the initial value of v.

Steady state Hopf bifurcations. Hopf bifurcations, $(x^* \in \mathbb{R}^n, v \in \mathbb{C}^n, \omega \in \mathbb{R})$ are determined from the following $2n+1$-dimensional partially complex (and by this fact more properly called a $3n+2$-dimensional) determining system using one free parameter.

$$\begin{aligned} 0 &= f(x^*, x^*, \ldots, x^*, \eta) \\ 0 &= \Delta(x^*, \eta, \mathrm{i}\omega)v \\ 0 &= c^{\mathrm{H}} v - 1 \end{aligned} \tag{19}$$

Periodic solutions. Periodic solutions are found as solutions $(u(s),\ s \in [0,1]; T \in \mathbb{R})$ of the following $(n(Ld+1)+1$-dimensional system with no free parameters.

$$\begin{aligned} \dot{u}(c_{i,j}) = Tf\left(u(c_{i,j}), u\left(\left[c_{i,j} - \frac{\tau_1}{T}\right|_{\text{ mod } [0,1]}\right), \ldots, u\left(\left[c_{i,j} - \frac{\tau_m}{T}\right|_{\text{ mod } [0,1]}\right)\right), \\ i = 0, \ldots, L-1,\ j = 1, \ldots, d \\ u(0) = u(1), \\ p(u) = 0. \end{aligned} \tag{20}$$

Here the notation $t|_{\text{ mod } [0,1]}$ refers to $t - \max\{k \in \mathbb{Z} : k \le t\}$, and p represents the integral phase condition

$$\int_0^1 \dot{u}(s)\Delta u(s)\mathrm{d}s = 0, \tag{21}$$

where u is the current solution and Δu its correction. The collocation points are obtained as

$$c_{i,j} = t_i + c_j(t_{i+1} - t_i),\ i = 0, \ldots, L-1,\ j = 1, \ldots, d,$$

from the interval points t_i, $i = 0, \ldots, L-1$ and the collocation parameters c_j, $j = 1, \ldots, d$. The profile u is discretized as a piecewise polynomial (see the manual). This representation has a discontinuous derivative at the interval points. If $c_{i,j}$ coincides with t_i the right derivative is taken in (20), if it coincides with t_{i+1} the left derivative is taken. In other words the derivative taken at $c_{i,j}$ is that of u restricted to $[t_i, t_{i+1}]$.

Connecting orbits. Connecting orbits can be found as solutions of the following determining system with $s^+ - s^- + 1$ free parameters, where s^+ and s^- denote the number of unstable eigenvalues of x^+ and x^- respectively.

$$\dot{u}(c_{i,j}) = Tf(u(c_{i,j}), u(c_{i,j} - \frac{\tau_1}{T}), \ldots, u(c_{i,j} - \frac{\tau_m}{T}), \eta) = 0, \quad (i = 0, \ldots, L-1,\ j = 1, \ldots, d)$$

$$
\begin{aligned}
u(\tilde{c}) =& x^- + \epsilon \sum_{k=1}^{s^-} \alpha_k v_k^- e^{\lambda_k^- T\tilde{c}}, \qquad \tilde{c} < 0 \\
0 =& f(x^-, x^-, \eta) \\
0 =& f(x^+, x^+, \eta) \\
0 =& \Delta(x^-, \lambda_k^-, \eta) v_k^- \\
0 =& c_k^H v_k^- - 1, \quad (k = 1, \ldots, s^-) \\
0 =& \Delta^H(x^+, \lambda_k^+, \eta) w_k^+ \\
0 =& d_k^H w_k^+ - 1, \quad (k = 1, \ldots, s^+) \\
0 =& w_k^{2H}(u(1) - x^+) + \sum_{i=1}^{G} g_i w_k^{+H} e^{-\lambda_k^+(\theta_i+\tau)} A_1(x^+, \eta) \left(u(1 + \frac{\theta_i}{T}) - x^+ \right), \quad (k = 1, \ldots, s^+) \\
u(0) =& x^- + \epsilon \sum_{i=1}^{s^-} \alpha_k v_k^- \\
1 =& \sum_{i=1}^{s^-} |\alpha_k|^2 \\
0 =& p(u, \eta)
\end{aligned}
\tag{22}
$$

Again, all arguments of u are taken modulo $[0, 1]$. We choose the same phase condition as for periodic solutions and similar normalization of v_k^- and $w + k^+$ as in (19).

Point method parameters. The point method parameters (see table 2) specify the following options.

- `newton_max_iterations`: maximum number of Newton iterations.
- `newton_nmon_iterations`: during a first phase of `newton_nmon_iterations`+1 Newton iterations the norm of the residual is allowed to increase. After these iterations, corrections are halted upon residual increase.
- `halting_accuracy`: corrections are halted when the norm of the last computed residual is less than or equal to `halting_accuracy` is reached.
- `minimal_accuracy`: a corrected point is accepted when the norm of the last computed residual is less than or equal to `minimal_accuracy`.
- `extra_condition`: this parameter is nonzero when extra conditions are provided in a routine bfsys_cond.m which should border the determining systems during corrections. The routine accepts the current point as input and produces an array of condition residuals and corresponding condition derivatives (as an array of point structures) as illustrated below (section 7.2).
- `print_residual_info`: when nonzero, the Newton iteration number and resulting norm of the residual are printed to the screen during corrections.

For periodic solutions and connecting orbits, the extra mesh parameters (see table 2) provide the following information.

- `phase_condition`: when nonzero the integral phase condition (21) is used.

- `collocation_parameters`: this parameter contains user given collocation parameters. When empty, Gauss-Legendre collocation points are chosen.

- `adapt_mesh_before_correct`: before correction and if the mesh inside the point is nonempty, adapt the mesh every `adapt_mesh_before_correct` points. E.g.: if zero, do not adapt; if one, adapt every point; if two adapt the points with odd point number.

- `adapt_mesh_after_correct`: similar to `adapt_mesh_before_correct` but adapt mesh after successful corrections and correct again.

7.2. Extra conditions

When correcting a point or computing a branch, it is possible to add one or more extra conditions and corresponding free parameters to the determining systems presented earlier. These extra conditions should be implemented using a function `sys_cond` and setting the method parameter `extra_condition` to 1 (cf. table 2). The function `sys_cond` accepts the current point as input and produces a residual and corresponding condition derivatives (as a point structure) per extra condition.

As an example, suppose we want to compute a branch of periodic solutions of system (10) subject to the following extra conditions

$$\begin{aligned} T &= 200, \\ 0 &= a_{12}^2 + a_{21}^2 - 1, \end{aligned} \tag{23}$$

that is, we wish to continue a branch with fixed period $T = 200$ and parameter dependence $a_{12}^2 + a_{21}^2 = 1$. The routine shown in Listing 3 implements these conditions by evaluating and returning each residual for the given point and the derivatives of the conditions w.r.t. all unknowns (that is, w.r.t. to all the components of the point structure).

```
function [resi,condi]=sys_cond(point)
% kappa beta a12 a21 tau1 tau2 tau_s
if point.kind=='psol'
  fix period at 200:
  resi(1)=point.period-200;
  % derivative of first condition wrt unknowns:
  condi(1)=p_axpy(0,point,[]);
  condi(1).period=1;
  % parameter condition:
  resi(2)=point.parameter(3)^2+point.parameter(4)^2-1;
  % derivative of second condition wrt unknowns:
  condi(2)=p_axpy(0,point,[]);
  condi(2).parameter(3)=2*point.parameter(3);
  condi(2).parameter(4)=2*point.parameter(4);
else
  error('SYS_COND: point is not psol.');
end
end
```

Listing 3. Implementation extra conditions (23) using a routine `sys_cond`.

7.3. Continuation

During continuation, a branch is extended by a combination of predictions and corrections. A new point is predicted based on previously computed points using secant prediction over an appropriate steplength. The prediction is then corrected using the determining systems (17), (18), (19), (20) or (22) bordered with a steplength condition which requires orthogonality of the correction to the secant vector. Hence one extra free parameter is necessary compared to the numbers mentioned in the previous section.

The following continuation and steplength determination strategy is used. If the last point was successfully computed, the steplength is multiplied with a given, constant factor greater than 1. If corrections diverged or if the corrected point was rejected because its accuracy was not acceptable, a new point is predicted, using linear interpolation, halfway between the last two successfully computed branch points. If the correction of this point succeeds, it is inserted in the point array of the branch (before the previously last computed point). If the correction of the interpolated point fails again, the last successfully computed branch point is rejected (for fear of branch switch) and the interpolation procedure is repeated between the (new) last two branch points. Hence, if, after a failure, the interpolation procedure succeeds, the steplength is approximately divided by a factor two. Test results indicate that this procedure is quite effective and proves an efficient alternative to using only (secant) extrapolation with steplength control. The reason for this is mainly that the secant extrapolation direction is not influenced by halving the steplength but it is by inserting a newly computed point in between the last two computed points.

For the description and the meaning of the continuation method parameters we refer to the manual.

7.4. Roots of the characteristic equation

Roots of the characteristic equation are approximated using a linear multi-step (LMS-) method applied to (2).

Consider the linear k-step formula

$$\sum_{j=0}^{k} \alpha_j y_{L+j} = h \sum_{j=0}^{k} \beta_j f_{L+j}. \tag{24}$$

Here, $\alpha_0 = 1$, h is a (fixed) step size and y_j presents the numerical approximation of $y(t)$ at the mesh point $t_j := jh$. The right hand side $f_j := f(y_j, \tilde{y}(t_j - \tau_1), \ldots, \tilde{y}(t_j - \tau_m))$ is computed using approximations $\tilde{y}(t_j - \tau_1)$ obtained from y_i in the past, $i < j$. In particular, the use of so-called Nordsieck interpolation, leads to

$$\tilde{y}(t_j + \epsilon h) = \sum_{l=-r}^{s} P_l(\epsilon) y_{j+l}, \ \epsilon \in [0, 1). \tag{25}$$

using

$$P_l(\epsilon) := \prod_{k=-r,\, k\neq l}^{s} \frac{\epsilon - k}{l - k}.$$

The resulting method is explicit whenever $\beta_0 = 0$ and $\min \tau_i > sh$. That is, y_{L+k} can then

directly be computed from (24) by evaluating

$$y_{L+k} = -\sum_{j=0}^{k-1} \alpha_j y_{L+j} + h \sum_{j=0}^{k} \beta_j f_{L+j}.$$

whose right hand side depends only on y_j, $j < L + k$.

For the linear variational equation (2) around a steady state solution $x^*(t) \equiv x^*$ we have

$$f_j = A_0 y_j + \sum_{i=0}^{m} A_i \tilde{y}(t_j - \tau_i) \tag{26}$$

where we have omitted the dependency of A_i on x^*. The stability of the difference scheme (24), (26) can be evaluated by setting $y_j = \mu^{j-L_{\min}}$, $j = L_{\min}, \ldots, L + k$ where $L_{\min}$ is the smallest index used, taking the determinant of (24) and computing the roots μ. If the roots of the polynomial in μ all have modulus smaller than unity, the trajectories of the LMS-method converge to zero. If roots exist with modulus greater than unity then trajectories exist which grow unbounded.

Since the LMS-method forms an approximation of the time integration operator over the time step h, so do the roots μ approximate the eigenvalues of $S(h, 0)$. The eigenvalues of $S(h, 0)$ are exponential transforms of the roots λ of the characteristic equation (5),

$$\mu = \exp(\lambda h).$$

Hence, once μ is found, λ can be extracted using,

$$\mathrm{Re}(\lambda) = \frac{\ln(|\mu|)}{h}. \tag{27}$$

The imaginary part of λ is found modulo π/h, using

$$\mathrm{Im}(\lambda) \equiv \frac{\arcsin(\frac{\mathrm{Im}(\mu)}{|\mu|})}{h} \ (\mathrm{mod}\ \frac{\pi}{h}). \tag{28}$$

For small h, $0 < h \ll 1$, the smallest representation in (28) is assumed the most accurate one (that is, we let arcsin map into $[-\pi/2, \pi/2]$).

The parameters r and s (from formula (25)) are chosen such that $r \leq s \leq r + 2$ (see [30]). The choice of h is based on the related heuristic outlined in [21].

Approximations for the rightmost roots λ obtained from the LMS-method using (27), (28) can be corrected using a Newton process on the system,

$$\begin{aligned} 0 &= \Delta(\lambda) v \\ 0 &= c^{\mathrm{T}} v - 1 \end{aligned} \tag{29}$$

A starting value for v is the eigenvector of $\Delta(\lambda)$ corresponding to its smallest eigenvalue (in modulus).

Note that the collection of successfully corrected roots presents more accurate yet less robust information than the set of uncorrected roots. Indeed, attraction domains of roots of equations like (29) can be very small and hence corrections may diverge, or different roots may be corrected to a single 'exact' root thereby missing part of the spectrum. The latter does not occur when computing the (full) spectrum of a discretization of $S(h, 0)$.

Stability information (the time step used, the number of Newton iterations, approximate roots

and corrected roots) is kept in a certain structure, see the manual. Also, see the manual for the stability method parameters.

7.5. Floquet multipliers

Floquet multipliers are computed as eigenvalues of the discretized time integration operator $S(T, 0)$. The discretization is obtained using the collocation equations (20) without the modulo operation (and without phase and periodicity condition). From this system a discrete, linear map is obtained between the variables presenting the segment $[-\tau/T, 0]$ and those presenting the segment $[-\tau/T + 1, 1]$. If these variables overlap, part of the map is just a time shift.

Stability information and approximations to the Floquet multipliers are kept in a certain structure, see the manual for the stability method parameters.

8. ILLUSTRATIVE EXAMPLE

In this section we present some results on bifurcation analysis of system (10) with several constant delays produced by DDE-BIFTOOL (see the manual and the demo `neuron` for detail).

After the user has defined the right-hand side of the system (see section 5.1), bifurcation analysis can be performed using the point and branch manipulation layers. System definitions files and the commands used in this demo are listed and extensively commented in the manual. Here we outline an illustrative ride-through for this example.

It is clear that (10) has a steady state solution $(x_1^*, x_2^*) = (0, 0)$ for all values of the parameters. We define a first steady state solution using the parameter values $\kappa = 0.5$, $\beta = -1$, $a_{12} = 1$, $a_{21} = 2.34$, $\tau_1 = \tau_2 = 0.2$ and $\tau_s = 1.5$. We set minimal_real_part to -2 (which means that roots are computed up to $\mathrm{Re}(\lambda) \geq -2$) and compute and plot stability of the steady state point, see figure 2 (left). The steady state has one unstable real mode. We will use the zero solution as

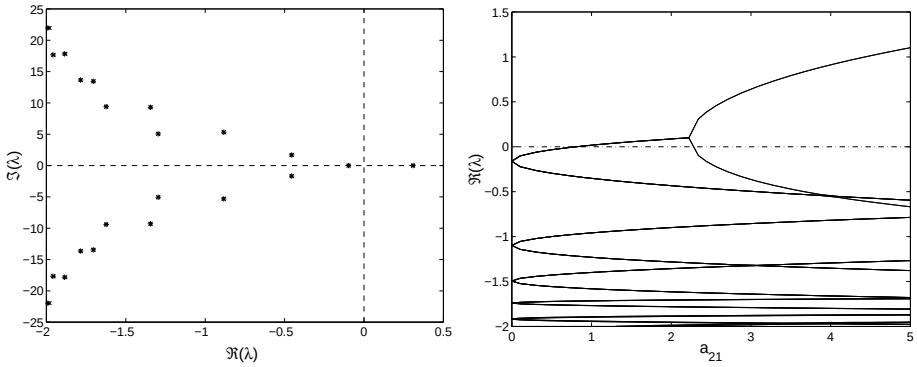

Fig. 2. Left: Approximated $(\times)$ and corrected $(*)$ roots of the characteristic equation of system (10) at its steady state solution $(x_1^*, x_2^*) = (0, 0)$. Real parts computed up to $\mathrm{Re}(\lambda) \geq -2$. Note that approximations $(\times)$ and corrections $(*)$ are nearly indistinguishable. Right: real parts of the approximated and corrected roots of the characteristic equation versus a_{21} along the branch of the steady state solution.

a first point to compute a branch of steady state solutions (not shown here, see the manual) and the stability along the branch. We plot the real part of the approximated and corrected roots of the characteristic equation along the branch, see figure 2 (right). This figure indicates the real part of the approximated and corrected roots of the characteristic equation along the branch.

From this figure alone it is not clear which real parts correspond to real roots respectively complex pairs of roots. For this it is useful to compare figures 2 (left and right). Notice the strange behaviour (coinciding of several complex pairs of roots) at $a_{21} = 0$. At this parameter value one

of the couplings between the neurons is broken. In fact, for $a_{21} = 0$, the evolution of the second component is independent of the evolution of the first.

Where lines cross the zero line, bifurcations occur. In particular, the Hopf bifurcation near $a_{21} \approx 0.8$. In order to follow a branch of Hopf bifurcations in the two parameter space (a_{21}, τ_s) we need two starting points. Hence we use the Hopf point already found and one perturbed in τ_s and corrected in a_{21}, to start on a branch of Hopf bifurcations. For the free parameters, a_{21} and τ_s, we provide suitable intervals, $a_{21} \in [0, 4]$ and $\tau_s \in [0, 10]$, and maximal stepsizes, 0.2 for a_{21} and 0.5 for τ_s and we continue the branch on both sides.

For this example, we do not change continuation method parameters, so that predictions and corrections are plotted during continuation. The final result is shown in figure 3 (left). At the top, the branch hits the boundary $\tau_s = 10$. To the right, however, it seemingly turned back onto itself. We compute and plot stability along the branch, see figure 4 (left).

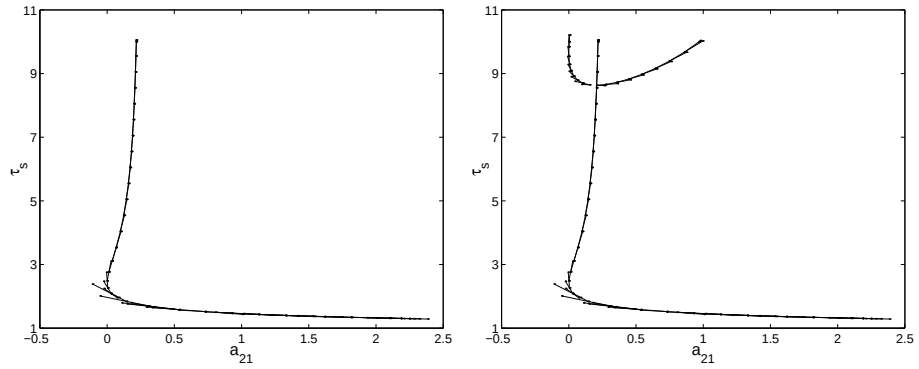

Fig. 3. Predictions and corrections in the (a_{21}, τ_s)-plane after computation of a first branch of Hopf bifurcations (left) and a second, intersecting branch of Hopf bifurcations (right).

We notice a double Hopf point on the left but nothing special at the right end, which could explain the observed turning of the branch. Plotting the frequency ω versus τ_s reveals what has happened, see figure 4 (right). For small τ_s, ω goes through zero, indicating the presence of a Bogdanov-Takens point. Using the second Hopf point we compute the intersecting branch of Hopf points depicted in figure 3 (right).

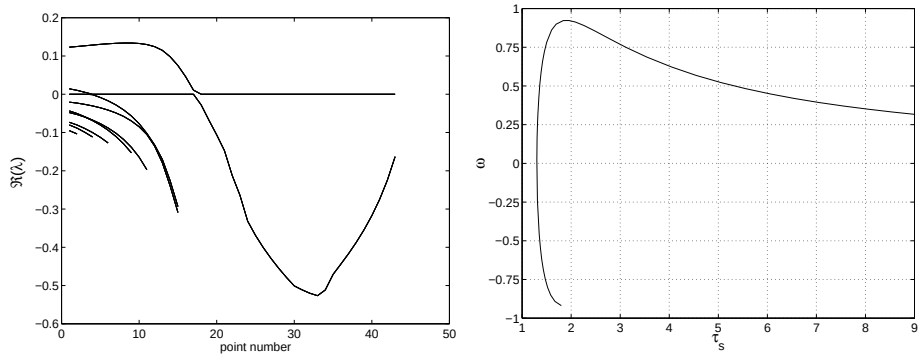

Fig. 4. Left: Real part of characteristic roots along the branch of Hopf bifurcations shown in figure 3 (left). Right: The frequency of the Hopf bifurcation along the same branch.

We use the first Hopf point we computed to construct a small amplitude $(1e-2)$ periodic solution on an equidistant mesh of 18 intervals with piecewise polynomial degree 3. The steplength condition used in the code returned ensures the branch switch from the Hopf to the

periodic solution as it avoids convergence of the amplitude to zero during corrections. Due to the presence of the steplength condition we also need to free one parameter, here a_{21}. The result, along with a degenerate periodic solution with amplitude zero is used to start on the emanating branch of periodic solutions, see figure 5 (left). We avoid adaptive mesh selection and an equidistant mesh is then automatically used which is kept fixed during continuation. Zooming shows erratic behaviour of the last computed branch points, shortly beyond a turning point, see figure 5 (right).

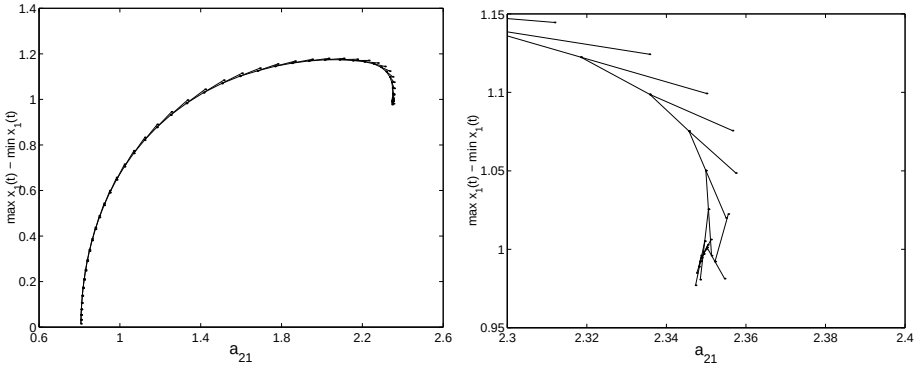

Fig. 5. Branch of periodic solutions emanating from a Hopf point (left). The branch turns at the far right and a zoom (right) indicates computational difficulties at the end.

Plotting some of the last solution profiles shows that smoothness and thus also accuracy are lost, see figure 6 (left). From a plot of the period along the branch we could suspect a homoclinic or heteroclinic bifurcation scenario, see figure 7 (left).

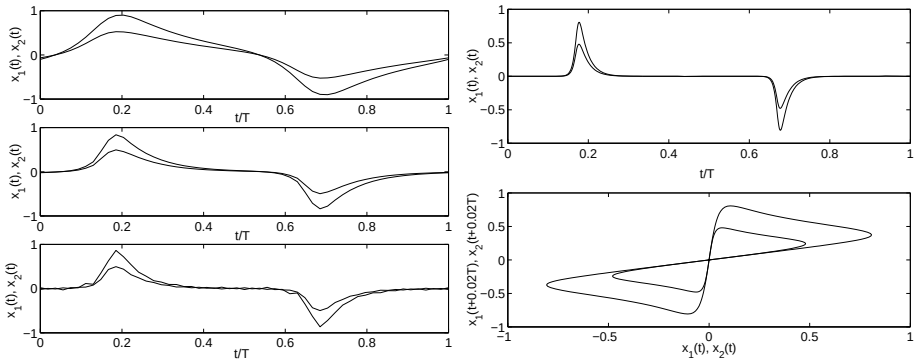

Fig. 6. Some solution profiles using equidistant meshes (left) and adapted meshes -(right) along the branch of periodic solutions shown in figure 5.

The result of computing and plotting stability (Floquet multipliers) just before and after the turning point is shown in figure 8. The second spectrum is clearly unstable.

First, we recompute a point on a refined, adapted mesh. Then we recompute the branch using adaptive mesh selection (with reinterpolation and additional corrections) after correcting every point, see figure 7 (right). Plotting of a point clearly shows the (double) homoclinic nature of the solutions, see figure 6 (right).

9. CONCLUDING COMMENTS

The first aim of DDE-BIFTOOL is to provide a portable, user-friendly tool for numerical bifurcation analysis of steady state solutions and periodic solutions of systems of delay

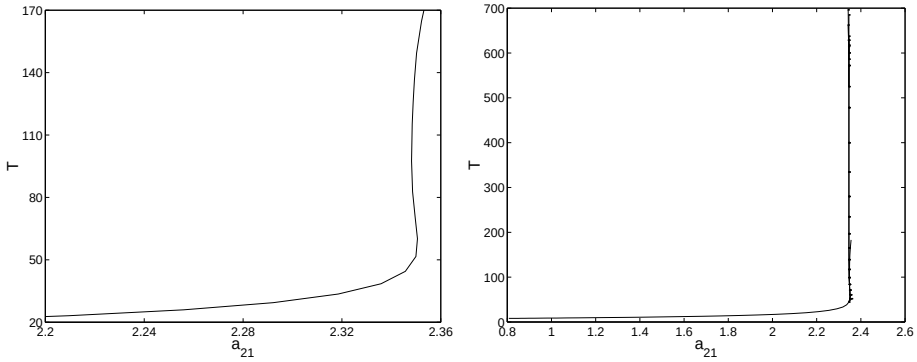

Fig. 7. Left: Period along the computed branch shown in figure 5. Right: Added period predictions and corrections during recalculations using adaptive mesh selection.

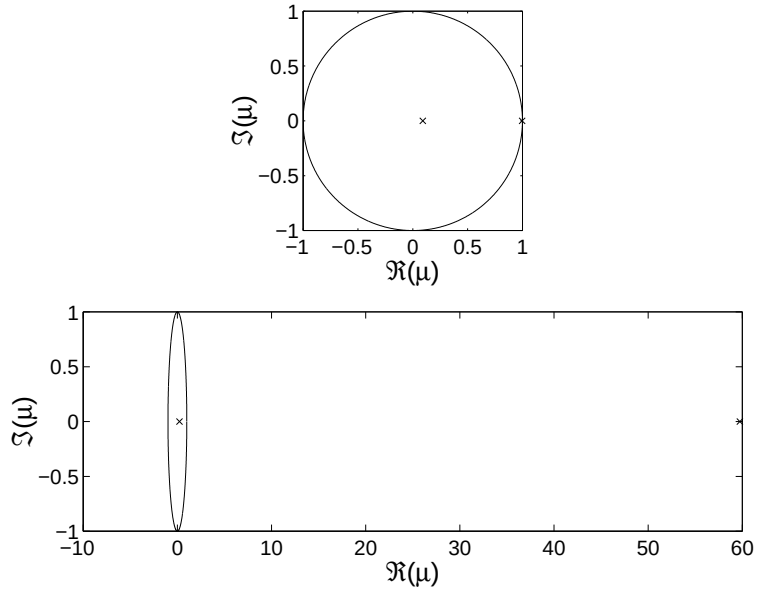

Fig. 8. Floquet multipliers for a periodic solutions before (top) and just after (bottom) the turning point visible in figure 5.

differential equations of the kinds (1) and (7). Part of this goal was fulfilled through choosing the portable, programmer-friendly environment offered by Matlab. Robustness with respect to the numerical approximation is achieved through automatic steplength selection in approximating the rightmost characteristic roots and through collocation using piecewise polynomials combined with adaptive mesh selection.

Although the package has been successfully tested on a number of realistic examples, a word of caution may be appropriate. First of all, the package is essentially a research code (hence we accept no reliability) in a quite unexplored area of current research. In our experience up to now, new examples did not fail to produce interesting theoretical questions (e.g., concerning homoclinic or heteroclinic solutions) many of which remain unsolved today. Unlike for ordinary differential equations, discretization of the state space is unavoidable during computations on delay equations. Hence the user of the package is strongly advised to investigate the effect of discretization using tests on different meshes and with different method parameters; and, if possible, to compare with analytical results and/or results obtained using simulation.

Although there are no ’hard’ limits programmed in the package (with respect to system and/or mesh sizes), the user will notice the rapidly increasing computation time for increasing

system dimension and mesh sizes. This is most notable in the stability and periodic solution computations. Indeed, eigenvalues are computed from large sparse matrices without exploiting sparseness and the Newton procedure for periodic solutions is implemented using direct methods. Nevertheless the current version is sufficient to perform bifurcation analysis of systems with reasonable properties in reasonable execution times. Furthermore, we hope future versions will include routines which scale better with the size of the problem.

Existing extensions

- Extension `debiftool_extra_psol` continues the three local codimension-one bifurcations of periodic orbits, the fold bifurcation, the period doubling and the torus bifurcation for constant and state-dependent delays.
- Extension `debiftool_extra_rotsym` continues relative equilibria and relative periodic orbits and their local codimension-one bifurcations for constant delays in systems with rotational symmetry (that is, there exists a matrix $A \in \mathbb{R}^{n\times n}$ such that $A^T = -A$ and $\exp(At)f(x_0,\ldots,x_m) = f(\exp(At)x_0,\ldots,\exp(At)x_m)$).
- Extension `debiftool_extra_nmfm` computes normal form coefficients of Hopf bifurcations, Hopf-Hopf interactions, generalized Hopf (Bautin) bifurcations, and zero-Hopf interactions (Gavrilov-Guckenheimer bifurcations) for equations with constant delays.

Possible future developments. These include:

- a graphical user interface;
- incorporation of the numerical core routines into a general continuation framework such as `Coco` [8] (which would permit the user to grow higher-dimensional solution families and wrap other continuation algorithms around the core DDE routines),
- the extension to other types of delay equations such as distributed delay and neutral functional differential equations. See also Barton et al [3] for a demonstration of how to extend DDE-BIFTOOL to neutral functional differential equations.
- determination of more normal-form coefficients to detect other co-dimension-two bifurcations.

DDE-BIFTOOL v. 2.03 is a result of the research project OT/98/16, funded by the Research Council K.U.Leuven; of the research project G.0270.00 funded by the Fund for Scientific Research - Flanders (Belgium) and of the research project IUAP P4/02 funded by the programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture. K. Engelborghs is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium). J. Sieber's contribution to the revision leading to version 3.0 was supported by EPSRC grant EP/J010820/1.

REFERENCES

1. Argyris J., Faust G., Haase M. *An Exploration of Chaos – An Introduction for Natural Scientists and Engineers*. North Holland Amsterdam, 1994.
2. Azbelev N.V., Maksimov V.P., Rakhmatullina L.F. *Introduction to the Theory of Functional Differential Equations*. Moscow: Nauka, 1991 (in Russ.).

3. Barton D.A.W., Krauskopf B., Wilson R.E. Collocation schemes for periodic solutions of neutral delay differential equations. *Journal of Difference Equations and Applications*. 2006. V. 12. No. 11. P. 1087–1101.
4. Bellman R., Cooke K.L. *Differential-Difference Equations*. Academic Press, 1963. (Mathematics in science and engineering. V. 6).
5. Breda D., Maset S., Vermiglio R. TRACE-DDE: a Tool for Robust Analysis and Characteristic Equations for Delay Differential Equations. In: *Topics in Time Delay Systems: Analysis, Algorithms, and Control*. Eds. J.J. Loiseau, W. Michiels, S.-I. Niculescu, R. Sipahi. New York: Springer, 2009. V. 388. P. 145–155. (Lecture Notes in Control and Information Sciences).
6. Chow S.-N., Hale J.K. *Methods of Bifurcation Theory*. Springer-Verlag, 1982.
7. Corwin S.P., Sarafyan D., Thompson S. DKLAG6: A Code Based on Continuously Imbedded Sixth Order Runge-Kutta Methods for the Solution of State Dependent Functional. *Applied Numerical Mathematics*. 1997. V. 24. No. 2–3. P. 319–330.
8. Dankowicz H., Schilder F. *Recipes for Continuation*. SIAM, 2013. (Computer Science and Engineering).
9. Dhooge A., Govaerts W., Kuznetsov Y.A. MatCont: A Matlab package for numerical bifurcation analysis of ODEs. *ACM Transactions on Mathematical Software*. 2003. V. 29. No. 2. P. 141–164.
10. Diekmann O., van Gils S.A., Verduyn Lunel S.M., Walther H.-O. *Delay Equations: Functional-, Complex-, and Nonlinear Analysis*. Springer-Verlag, 1995. (Applied Mathematical Sciences. V. 110).
11. Doedel E.J. Lecture notes on numerical analysis of nonlinear equations. In: *Numerical Continuation Methods for Dynamical Systems: Path following and boundary value problems*. Eds. Krauskopf B., Osinga H.M., Galán-Vioque J. Dordrecht: Springer-Verlag, 2007. P. 1–49.
12. Doedel E.J., Champneys A.R., Fairgrieve T.F., Kuznetsov Y.A., Sandstede B., Wang X. *AUTO97*: Continuation and bifurcation software for ordinary differential equations; available by FTP from ftp.cs.concordia.ca in directory pub/doedel/auto.
13. Driver R.D. *Ordinary and Delay Differential Equations*. Springer-Verlag, 1977.(Applied Mathematical Science. V. 20).
14. El'sgol'ts L.E., Norkin S.B. *Introduction to the Theory and Application of Differential Equations with Deviating Arguments*. Academic Press, 1973. (Mathematics in science and engineering. V. 105).
15. Engelborghs K. *Numerical Bifurcation Analysis of Delay Differential Equations*: PhD thesis. Leuven, Belgium: Department of Computer Science, Katholieke Universiteit Leuven, 2000.
16. Engelborghs K., Doedel E. Stability of piecewise polynomial collocation for computing periodic solutions of delay differential equations. *Numerische Mathematik*. 2002. V. 91. No. 4. P. 627–648.
17. Engelborghs K., Luzyanina T., In t́ Hout K.J., Roose D. Collocation methods for the computation of periodic solutions of delay differential equations. *SIAM J. Sci. Comput.* 2000. V. 22. P. 1593-1609.
18. Engelborghs K., Luzyanina T., Roose D. Numerical bifurcation analysis of delay differential equations using DDE-BIFTOOL. *ACM Transactions on Mathematical Software*. 2002. V. 28. No. 1. P. 1–21.
19. Engelborghs K., Luzyanina T., Samaey G. *DDE-BIFTOOL v.2.00: a Matlab package for bifurcation analysis of delay differential equations*: Report TW 330. Katholieke Universiteit Leuven, 2001.
20. Engelborghs K., Roose D. Numerical computation of stability and detection of Hopf

bifurcations of steady state solutions of delay differential equations. *Advances in Computational Mathematics*. 1999. V. 10. No. 3–4. P. 271–289.
21. Engelborghs K., Roose D. On stability of LMS-methods and characteristic roots of delay differential equations. *SIAM J. Num. Analysis*. 2002. V. 40. No. 2. P. 629–650.
22. Enright W.H., Hayashi H. A delay differential equation solver based on a continuous Runge-Kutta method with defect control. *Numer. Algorithms*. 1997. V. 16. P. 349–364.
23. Ermentrout B. *XPPAUT 3.91 - The differential equations tool*. Pittsburgh: University of Pittsburgh, 1998. URL: http://www.pitt.edu/∼phase/.
24. Govaerts W.J.F. *Numerical Methods for Bifurcations of Dynamical Equilibria*. SIAM, 2000. (Miscellaneous Titles in Applied Mathematics Series).
25. Guckenheimer J., Holmes P. *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields*. New York: Springer-Verlag, 1983.
26. Guglielmi N., Hairer E. Stiff delay equations. *Scholarpedia*. 2007. V. 2. No. 11. P. 2850.
27. Hale J.K. *Theory of Functional Differential Equations*. Springer-Verlag, 1977. (Applied Mathematical Sciences. V. 3).
28. Hale J.K., Verduyn Lunel S.M. *Introduction to Functional Differential Equations*. Springer-Verlag, 1993. (Applied Mathematical Sciences. V. 99).
29. Hartung F., Krisztin T., Walther H.-O. , Wu J. Functional differential equations with state-dependent delays: Theory and applications. In: *Handbook of Differential Equations: Ordinary Differential Equations*. Eds. P. Drábek and A. Cañada and A. Fonda. North-Holland, 2006. V.3. Chapter 5. P. 435–545.
30. Hong-Jiong T., Jiao-Xun K. The numerical stability of linear multistep methods for delay differential equations with many delays. *SIAM Journal of Numerical Analysis*. 1996. V. 33. No. 3. P. 883–889.
31. The MathWorks Inc. *MATLAB*. Natick, Massachusetts, United States
32. Kolmanovskii V., Myshkis A. *Applied Theory of Functional Differential Equations*. Kluwer Academic Publishers, 1992. (Mathematics and Its Applications. V. 85).
33. Kolmanovskii V.B., Myshkis A. *Introduction to the theory and application of functional differential equations*. Kluwer Academic Publishers, 1999. (Mathematics and its applications. V. 463).
34. Kolmanovskii V.B., Nosov V.R. *Stability of functional differential equations*. Academic Press, 1986. (Mathematics in Science and Engineering. V. 180).
35. Kuznetsov Y.A. *Elements of Applied Bifurcation Theory*. New York: Springer-Verlag, 2004. (Applied Mathematical Sciences. V. 112).
36. Luzyanina T., Engelborghs K., Roose D. Numerical bifurcation analysis of differential equations with state-dependent delay. *Internat. J. Bifur. Chaos*. 2001. V. 11. No. 3. P. 737–754.
37. Luzyanina T., Roose D. Numerical stability analysis and computation of Hopf bifurcation points for delay differential equations. *Journal of Computational and Applied Mathematics*. 1996. V. 72. P. 379–392.
38. Mallet-Paret J., Nussbaum R.D. Stability of periodic solutions of state-dependent delay-differential equations. *Journal of Differential Equations*. 2011. V. 250. No. 11. P. 4085–4103.
39. Paul C.A.H. *A user-guide to Archi - an explicit Runge-Kutta code for solving delay and neutral differential equations*: Technical Report 283. The University of Manchester, Manchester Center for Computational Mathematics, 1997.
40. Roose D., Szalai R. Continuation and bifurcation analysis of delay differential equations. In: *Numerical Continuation Methods for Dynamical Systems: Path following and boundary value problems*. Eds. Krauskopf B., Osinga H. M., Galán-Vioque J. Dordrecht:

Springer-Verlag, 2007. P. 51–75.
41. Samaey G., Engelborghs K., Roose D. *Numerical computation of connecting orbits in delay differential equations*: Report TW 329. Leuven, Belgium: Department of Computer Science, K.U.Leuven, 2001. 20 p. URL: http://www.cs.kuleuven.ac.be/publicaties/rapporten/tw/TW329.abs.html
42. Seydel R. *Practical Bifurcation and Stability Analysis – From Equilibrium to Chaos*. Berlin: Springer-Verlag, 1994. (Interdisciplinary Applied Mathematics. V. 5).
43. Shampine L.F., Thompson S. *Solving delay differential equations with dde23*. Submitted, 2000.
44. Shayer L.P., Campbell S.A. Stability, bifurcation and multistability in a system of two coupled neurons with multiple time delays. *SIAM J. Applied Mathematics*. 2000. V. 61. No. 2. P. 673–700.
45. Sieber J. Finding periodic orbits in state-dependent delay differential equations as roots of algebraic equations. *Discrete and Continuous Dynamical Systems A*. 2012. V. 32. No. 8. P. 2607–2651.
46. Szalai R., Stépán G., Hogan S.J. Continuation of bifurcations in periodic delay differential equations using characteristic matrices. *SIAM Journal on Scientific Computing*. 2006. V. 28. No. 4. P. 1301–1317.

Accepted 21.11.2017.
Published 13.12.2017.