MATCONT: A MATLAB Package for Numerical Bifurcation Analysis of ODEs

A. DHOOGE and W. GOVAERTS
Ghent University
and
YU. A. KUZNETSOV
Utrecht University

MATCONT is a graphical MATLAB software package for the interactive numerical study of dynamical systems. It allows one to compute curves of equilibria, limit points, Hopf points, limit cycles, period doubling bifurcation points of limit cycles, and fold bifurcation points of limit cycles. All curves are computed by the same function that implements a prediction-correction continuation algorithm based on the Moore-Penrose matrix pseudo-inverse. The continuation of bifurcation points of equilibria and limit cycles is based on bordering methods and minimally extended systems. Hence no additional unknowns such as singular vectors and eigenvectors are used and no artificial sparsity in the systems is created. The sparsity of the discretized systems for the computation of limit cycles and their bifurcation points is exploited by using the standard Matlab sparse matrix methods. The MATLAB environment makes the standard MATLAB Ordinary Differential Equations (ODE) Suite interactively available and provides computational and visualization tools; it also eliminates the compilation stage and so makes installation straightforward. Compared to other packages such as AUTO and CONTENT, adding a new type of curves is easy in the MATLAB environment. We illustrate this by a detailed description of the limit point curve type.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Algorithm design and analysis; User interfaces; G.1.7 [**Numerical Analysis**]: Ordinary differential equations; J.2 [**Physical Sciences and Engineering**]: Mathematics and Statistics

General Terms: Design

Additional Key Words and Phrases: Dynamical system, bifurcation, numerical continuation

1. INTRODUCTION

1.1 Some Mathematical Prerequisites

We consider generic parameterized autonomous ordinary differential equations (ODEs) of the form

$$\frac{dx}{dt} \equiv \dot{x} = f(x, \alpha), \tag{1}$$

Author's addresses: A Dhooge and W. Govaerts, Department of Applied Mathematics and Computer Science, University of Ghent, Krijgslaan 281-S9, B-9000 Ghent, Belgium; email: {Annick.Dhooge,Willy.Govaerts@rug.ac.be; Yu. A. Kuznetsov, Mathematisch Instituut, Universiteit Utrecht, Boedapestlaan 6, 3584 CD Utrecht, The Netherlands, and Institute of Mathematical Problems in Biology, Pushchino, Moscow Region, 142290 Russia; email: kuznetsov@math.uu.nl.

where $x \in \mathbb{R}^n$ is the vector of *state variables*, $\alpha \in \mathbb{R}^m$ represents *parameters*, and $f(x, \alpha) \in \mathbb{R}^n$. Examples of systems of the form (1) are ubiquitous in mathematical models in physics, engineering, chemistry, biology, economics, finance, etc.

The image of the definition domain of a solution $t \mapsto x(t)$ of (1) is called an *orbit*. The simplest orbits are the *equilibria*, that is, solutions of the equation

$$f(x, \alpha) = 0. \tag{2}$$

As is well known, an equilibrium $x = x_0$ is asymptotically stable at $\alpha = \alpha_0$ if all eigenvalues of the Jacobian matrix $f_x(x_0, \alpha_0)$ have a strictly negative real part, and is unstable if there is at least one eigenvalue with a strictly positive real part. In generic one-parameter problems, eigenvalues on the imaginary axis appear in two ways: as a simple zero eigenvalue, or as a conjugate pair $\pm i\omega$, $\omega > 0$, of simple pure imaginary eigenvalues. The first case corresponds to a *fold*, where two solutions coalesce and annihilate each other under parameter variation. The second case corresponds to a *Hopf bifurcation*, from which periodic solutions emerge.

Periodic solutions are solutions for which $x(T) = x(0)$, for some number $T > 0$. The minimal such T is called the *period*. Nontrivial periodic solutions give rise to *closed orbits* (*cycles*) in the state space. The monodromy matrix is the linearized T-shift along orbits of Equation (1), evaluated at a point of the periodic solution. The eigenvalues of this matrix are the *Floquet multipliers* of the periodic solution [Guckenheimer and Holmes 1983; Kuznetsov 1998].

A periodic solution always has a multiplier equal to 1. If all other multipliers are strictly inside the unit circle in the complex plane, then the periodic solution is asymptotically stable. If at least one multiplier has modulus greater than 1, then the periodic solution is unstable. Three generic bifurcations, determined by the monodromy matrix, can occur along a one-parameter curve of periodic solutions, namely, the *fold*, the *period-doubling* (or *flip*) bifurcation, and the *torus* (or *Neimark-Sacker*) bifurcation. At a fold, the multiplier 1 has algebraic multiplicity 2 and geometric multiplicity 1. Generically, a fold corresponds to a point on the periodic solution branch where the curve turns with respect to the free problem parameter. At a period-doubling bifurcation point, there is a simple multiplier equal to -1. Generically this indicates a *period doubling* of the periodic solution, that is, there are nearby periodic solutions of approximately double period. At a torus bifurcation, there is a simple conjugate pair of complex eigenvalues with modulus 1. Generically this corresponds to a bifurcation to an invariant torus, on which the flow contains periodic or quasiperiodic motion.

Curves of equilibria, periodic orbits, etc. can be computed using numerical continuation which is the numerical pendant of homotopy methods. We consider a general system of nonlinear equations

$$F(X) = 0, \tag{3}$$

where $F(X) \in \mathbb{R}^N$ is a smooth function of $X \in \mathbb{R}^{N+1}$. If X_0 is a solution of Equation (3) and if the Jacobian matrix $F_X(X_0)$ has full rank N, then, by the Implicit Function Theorem, Equation (3) has a curve of solutions that passes through X_0. Numerical continuation produces a sequence of points on this curve

to within a given accuracy. Most continuation packages use a tangent predictor and a Newton-type corrector at each step, however, with slightly different corrector algorithms, and therefore compute different points on the curve. The convergence properties of the iteration are used to update the choice of the stepsize. For background information, see Keller [1977], Doedel et al. [1991], Allgower and Georg [1996], Beyn et al. [2002], Chapter 10 in Kuznetsov [1998], and Chapter 2 in Govaerts [2000]. More details on a particular continuation method implemented in MATCONT are given in Section 2.

To detect and accurately locate bifurcations along a computed curve, we need *test functions* that have a regular zero at such bifurcation points.

For example, a possible test function for the fold bifurcation along a curve of equilibria is the determinant function $\det(f_x(x,\alpha))$. A test function for the Hopf bifurcation is the determinant of the bialternate product matrix $2f_x \odot I_n$. If A, B are $n \times n$ matrices, then $A \odot B$ is an $m \times m$ matrix where $m = n(n-1)/2$. In the case $B = I_n$, this square matrix with dimension $\frac{n(n-1)}{2}$ has only $n(n-1)(2n-3)/2$ functionally nonzero entries, so for large n it is rather sparse. For details see Kuznetsov [1998] or Govaerts [2000].

If an additional parameter is freed, then we can compute curves of the detected bifurcation type. For this purpose, we need *defining systems*, that is, systems of equations whose regular solutions provide bifurcation points. The test function itself is not necessarily a good choice for continuation. For example, the determinant function suffers from ill-scaling, and its symbolic derivatives are hard to compute. In MATCONT, curves are computed by *minimally extended defining systems*. For example, the system for fold curves has the form

$$\begin{cases} f(x,\alpha) = 0, \\ g(x,\alpha) = 0, \end{cases} \tag{4}$$

where g is obtained by solving

$$\begin{pmatrix} f_x(x,\alpha) & b \\ c^T & d \end{pmatrix} \begin{pmatrix} v \\ g \end{pmatrix} = \begin{pmatrix} 0_n \\ 1 \end{pmatrix}, \tag{5}$$

and $b, c \in \mathbb{R}^n, d \in \mathbb{R}$ are chosen such that the matrix in Equation (5) is nonsingular. An advantage of this method is that the derivatives of g with respect to x and α can be obtained easily from the derivatives of f_x. The method also avoids the scaling problems [Govaerts 2000, §4.1.2].

1.2 Survey of Software

Three basic methods are used to study dynamical systems in applied modeling:

— simulation, including computation of Lyapunov exponents and dimensional and spectral characteristics;

— continuation of special solutions and their stability boundaries: computing one- or multiparameter families of equilibria [Henderson 2000], periodic or homoclinic orbits, using pathfollowing techniques and detecting and continuing critical (bifurcation) parameter values at which the system dynamic changes qualitatively; and

— normal form analysis: computing coefficients of reduced canonical equations (normal forms) near critical parameter values to predict which solutions and bifurcations are present locally to start their global continuation; understood broadly, this includes averaging and multiple scaling techniques.

There are several interactive software packages for analysis of dynamical systems defined by ODEs. The most widely used are

(1) AUTO86/97 [Doedel et al. 1997]. Runs on UNIX workstations under X-Windows and has a rudimentary graphical user interface (GUI). Supports continuation of equilibrium and periodic and homoclinic solutions and their bifurcations. Computationally advanced but very difficult to use; does not provide any normal form analysis. The most recent version AUTO2000 is written in C and has a command language interface based on the scripting language Python.
(2) DSTOOL [Back et al. 1992]. Runs on UNIX workstations under X-Windows, has a user-friendly tcl/tk-GUI and allows to vizualize computed data with GEOMVIEW. Supports simulation and (limited) continuation analysis. Computes one-dimensional invariant manifolds of fixed points. Has an extensive documentation that is both developer- and user-oriented. Possible, but hard, to extend.
(3) LOCBIF [Khibnik et al. 1993]. Runs as a DOS-application under MS-Windows, has a simple GUI. Supports continuation of equilibrium and periodic solutions, as well as the normal form analysis of the simplest bifurcations of equilibria. Has many restrictions on the model complexity and a closed architecture.
(4) CONTENT [Kuznetsov and Levitin 1995–1997]. Runs on UNIX workstations under X-Windows with (Open)Motif and on PCs under Windows-95/NT. Has a user-friendly GUI and an on-line hypertext help. Supports the continuation of equilibrium and periodic solutions and their bifurcations, as well as the normal form analysis of many bifurcations of equilibria and fixed points. Allows one to use a special linear algebra C-library for each solution type. Uses a very specific format to store the systems and results of computations. Automatically generates partial derivatives of the system right-hand sides. Possible, but hard, to extend.
(5) XPPAUT [Ermentrout 2002]. Runs on UNIX workstations, Windows, and Mac OSX. Has a GUI with animation. Allows one to simulate ODEs, discontinuous differential equations, delay equations and differential algebraic equations, some BVPs and PDEs. Uses AUTO numerics for the continuation of equilibrium and periodic solutions and their bifurcations.

These packages are mostly used at universities (in mathematics, physics, biology, and other departments). In industry, their usage is limited to chemical engineering and some aerospace research. The main reason for this is that all of these packages do not fit well into the standard engineering software environment. No one of these packages covers the whole range of solution types, while data exchange between them is practically impossible due to individual data

formats. They do not represent the results of the analysis in a form suitable for standard control, identification, and visualization software.

We note that several existing packages use MATLAB in dynamical system computations. Arnold and Polking [1999] described the use of simulation techniques and invariant curve computations in MATLAB, and provided a software tool PPLANE [Polking 1997–2003] for two-dimensional vector fields. Choe and Guckenheimer [2000] described guidelines for building an interface between MATLAB and dynamic tools, mainly oriented toward the use of AUTO. De Feo [2000] provided a tool MPLAUT to visualize the output of AUTO in MATLAB. Engelborghs et al. [2002] described a nongraphical but very extensive MATLAB package DDE-BIFTOOL to compute bifurcations of delay differential equations.

MATCONT is free MATLAB software available for download at

`http://allserv.rug.ac.be/~ajdhooge/research.html.`

The aim of the MATLAB software package MATCONT is to provide an interactive environment primarily designed for the continuation and normal form analysis of dynamical systems. This analysis is complementary to the simulation of the systems (which is also included in the package) and allows for more comprehensive understanding of their dynamics; it can be used in their identification, control, and optimization.

MATCONT implements a *starter-generator-processor* technique to compute different solutions to a dynamical system and to switch between them. Its computational kernel supports numerical continuation of regular curves implicitly specified by *defining functions* in a continuation space. It monitors scalar test functions, along a curve, and detects and locates critical parameter values, where these functions change sign. Such bifurcation points are processed using normal form computation to generate initial data for the continuation of other solution curves. All these subroutines are programmed in MATLAB in terms of the system right-hand sides and their partial derivatives.

MATCONT is a package under development that started with two master's theses [Riet 2000; Mestrom 2002]. In many respects it is similar to CONTENT, which can be considered as a prototype for MATCONT. However, MATCONT is being completely redesigned and reimplemented to exploit the power of MATLAB. It is being developed in parallel with the Continuation Toolbox CL_MATCONT [Dhooge et al. 2000–2002], a package of routines that can be used from the MATLAB command line. We note that CL_MATCONT can be used in Matlab 5.3 while MATCONT requires at least Matlab 6. CL_MATCONT is more general in the sense that it can also be used for other purposes, for example, the continuation of solutions to parameterized partial differential equations (PDEs) (upon a discretization).

The following functions are supported by the present version of MATCONT:

— continuation of equilibrium and periodic solutions with respect to a control parameter;
— detection of fold, Hopf, and branching points on curves of equilibria;
— normal form analysis of fold and Hopf equilibrium bifurcations;
— branch switching at equilibrium branch points;

—continuation of fold and Hopf equilibrium bifurcations in two control parameters;
—detection of all codim 2 equilibrium bifurcations (cusp, Bogdanov-Takens, generalized Hopf, zero-Hopf, and double Hopf) on fold and Hopf curves;
—detection of fold, flip, and torus bifurcations of periodic solutions;
—continuation of the period doubled orbit in a flip bifurcation point; and
—continuation of flip, fold, and torus (Neimark–Sacker) bifurcations of periodic orbits in two control parameters.

It is envisaged to include in the next versions:

—normal form analysis of all codim 2 equilibrium bifurcations;
—computation of one-dimensional invariant manifolds of equilibria;
—branch switching at cycle bifurcation points; and
—location and numerical continuation of orbits homoclinic to equilibria.

The GUI of MATCONT supports the following functions:

(1) specification of the initial solution to start the continuation (either by selecting a previously computed solution or by on-line specification of a new solution);
(2) selecting the solution type to continue;
(3) setting and modifying numerical parameters of the computation method specific to the selected initial point and solution type;
(4) activating detection of possible bifurcations and user-defined functions;
(5) starting computation/browsing, allowing for termination and pausing;
(6) manipulating the database of computed results by deleting or renaming the output files of the computations; and
(7) setting options for graphic windows to represent the computed solution during and after the computation.

Context-dependent help on supported solutions and used algorithms is an aim for future work.

2. NUMERICAL CONTINUATION IN MATCONT

All curves in MATCONT are computed by the same MATLAB-function that implements a prediction-correction continuation algorithm based on the Moore-Penrose matrix pseudoinverse.

Let A be an $N \times (N+1)$ matrix with rank $J = N$. Recall that its *Moore-Penrose pseudoinverse* is an $(N+1) \times N$ matrix A^+ defined by the formula

$$A^+ = A^T(AA^T)^{-1}.$$

To compute $A^+b,\ b \in \mathbb{R}^N$, efficiently, set up the system for $x \in \mathbb{R}^{N+1}$:

$$\begin{cases} Ax &= b, \\ v^T x &= 0, \end{cases} \tag{6}$$

where $v \in \mathbb{R}^{N+1}$ is such that $Av = 0$. Then $x = A^+b$ is a solution to this system, since $AA^+b = b,\ v^T A^+b = 0$.

Consider now an implicitly defined curve

$$F(x) = 0, \tag{7}$$

where the map $F : \mathbb{R}^{N+1} \to \mathbb{R}^N$ is smooth. A tangent vector $v^{(i)} \in \mathbb{R}^{N+1}$ to this curve at point $x^{(i)} \in \mathbb{R}^{N+1}$ satisfies

$$F_x(x^{(i)})v^{(i)} = 0.$$

Point $x^{(i)}$ is called *regular* if $v^{(i)}$ is unique (up to scaling). To compute regular points on the curve and their associated tangent vectors, a predictor-corrector method is implemented in MATCONT.

Prediction: Suppose that a point $x^{(i)}$ on the curve (or sufficently close to it) is found. The prediction for the next point is tangential:

$$X^0 = x^{(i)} + h_i v^{(i)}.$$

For the next point $x^{(i+1)} \in \mathbb{R}^{N+1}$ on the curve, one could solve

$$\begin{cases} F(x) &= 0, \\ v^T(x - X^0) &= 0, \end{cases}$$

where $v \in \mathbb{R}^{N+1}$ satisfies $F_x(x)v = 0$ (the solution minimizes the distance between X^0 and the curve described by Equation (7)). The linearization of this system about X^0 is

$$\begin{cases} F_x(X^0)(x - X^0) &= 0, \\ (V^0)^T(x - X^0) &= 0, \end{cases}$$

where $F_x(X^0)V^0 = 0$. Thus

$$x = X^0 - F_x^+(X^0)F(X^0),$$

leading to the so called *Moore-Penrose corrections*:

$$X^{k+1} = X^k - F_x^+(X^k)f(X^k),\ k = 0, 1, 2, \ldots$$

(see Figure 1). Note that $V^k \in \mathbb{R}^{N+1}$ satisfying $F_x(X^k)V^k = 0$ can be used to compute $F_x^+(X^k)$ efficiently with the help of Equation (6). However, V^k is unknown and is approximated in MATCONT by a vector V^k satisfying

$$F_x(X^{k-1})V^k = 0,\quad V^0 = v^{(i)}.$$

This yields the correction algorithm shown in Figure 2.

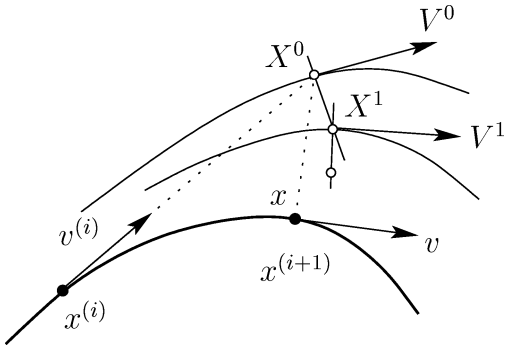

Fig. 1. Moore-Penrose corrections.

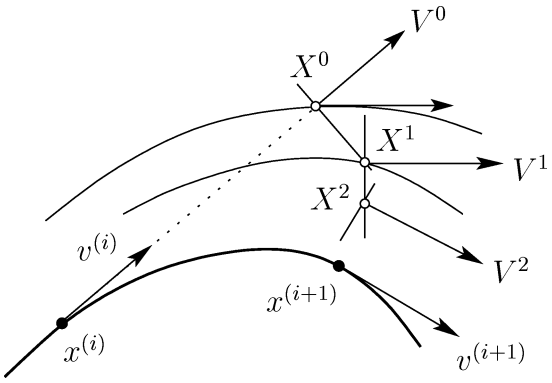

Fig. 2. Correction algorithm implemented in MATCONT.

Corrections: Iterate for $k = 0, 1, 2, \ldots, k_{max}$

$$\begin{aligned} A &= F_x(X^k), \quad B = \begin{pmatrix} A \\ V^{kT} \end{pmatrix}, \\ R &= \begin{pmatrix} AV^k \\ 0 \end{pmatrix}, \quad Q = \begin{pmatrix} F(X^k) \\ 0 \end{pmatrix}, \\ W &= V^k - B^{-1}R, \quad V^{k+1} = \frac{W}{\|W\|}, \\ X^{k+1} &= X^k - B^{-1}Q. \end{aligned}$$

If $\|F(X^k)\| < \varepsilon_f$ and $\|X^{k+1} - X^k\| < \varepsilon_x$, then

$$x^{(i+1)} = X^{k+1}, \quad v^{(i+1)} = V^{k+1}.$$

Here $\varepsilon_x, \varepsilon_f$ are convergence tolerances and k_{max} is the maximal number of corrections allowed. One can prove that thus defined corrections converge, if $x^{(i)}$ is a regular point on the curve and h_i is sufficiently small.

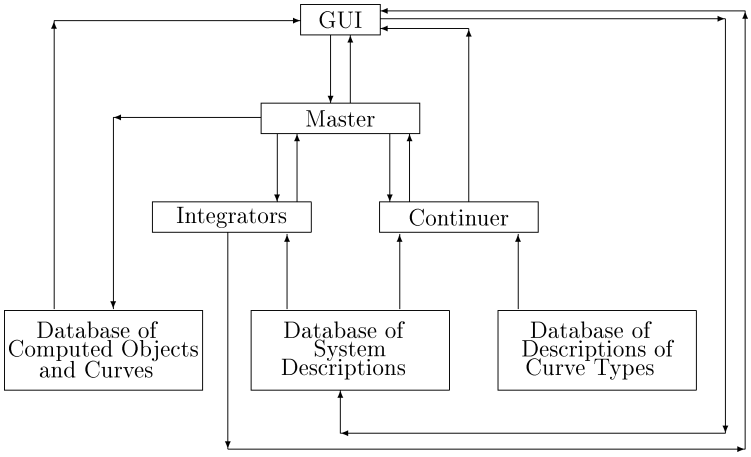

Fig. 3. Information flow in MATCONT.

Stepsize control: The next stepsize h_{i+1} is convergence-dependent. If the corrections converge, denote by k the number of iterations. Then the new stepsize is selected as

$$h_{i+1} = \begin{cases} Ch_i & \text{if not converged,} \\ ch_i & \text{if converged and } k < k_{thr}, \\ h_i & \text{otherwise,} \end{cases}$$

where $c < 1 < C$ and $k_{thr} < k_{max}$ are certain constants.

3. DATA FLOW AND DATA STRUCTURES IN MATCONT

Figure 3 presents a sketch of the information flow in MATCONT. The package has three databases. The user (GUI) can write directly to one of them, namely, the database of System Descriptions. Here one can introduce new dynamical systems or make changes in existing ones. One can read directly from the database of Computed Objects and Curves where one can study the computed results, make plots, print them out, etc. The third database, Descriptions of Curve Types is protected; the user has no direct access to it.

The computational work is performed either by the Integrators, which are the standard MATLAB ODE solvers [Shampine and Reichelt 1997] or by the Continuer, which is the general-purpose continuation algorithm described in Section 2. Finally, the most active part of the package is the Master program, which interacts in both directions with the GUI, the Continuer, and Integrators. Also, it writes the output of the computations to the database of Computed Objects and Curves. To make the environment fully interactive, the Continuer and the Integrators also write directly to the GUI. This slows the computations down somewhat; on the other hand, it gives the user more control, for example, to interrupt the computations.

The root directory of MATCONT is the location of the Master program *matcont.m*. The other files are kept in a number of separate directories which

include at least the following three general-purpose directories:

—Continuer,
—GUI, and
—Systems.

Furthermore, there are directories specific to each type of curve that can be computed (except Orbits). These include at least the following:

—Equilibrium,
—LimitPoint,
—Hopf,
—LimitCycle,
—PeriodDoubling, and
—LimitPointCycle.

(The list is growing.)

The fundamental global variables in the master program *matcont.m* are the MATLAB structures `gds` and `cds`. The structure `gds` contains the information concerning all windows open at a certain time and collects the information fed into the system by the GUI. `cds` is needed in the GUI to store a computed curve from whose points a new curve is to be started. Both `gds` and `cds` contain fields in which computational options such as `Increment`, `MaxNumPoints`, ..., are stored.

At the start of a new system, `gds.options` is empty. At the start of the computation of a curve, the continuer presents a Starter window and a Continuer window. The prefilled values in the Starter window are the default values stored internally in the Continuer. On the other hand, the prefilled values in the Continuer window are those that are given in `gds` if available. If they are not available in `gds`, then default values are proposed. The user now has the opportunity to change the fields in the Starter and Continuer windows. All changes will be incorporated in the `gds` fields.

As a consequence, if the user afterwards computes another curve, then the system will remember the values input by the user in the Continuer window, but not those in the Starter window.

The fields `gds.options.backward` and `cds.options.backward` behave in a somewhat different way because there is no effective default. The user chooses the direction of computing by clicking the "compute forward/backward" button in the MATCONT Main window and this effectively starts the continuation of the curve.

The continuer now copies `gds.options` to `cds.options` and looks if every field needed in `cds.options` is filled. If not, the continuer enters the default value. A list of the fields and main subfields of `gds` and `cds` in a typical situation is given in Appendices A and B.

The other global variables in MATCONT are specific to certain types of computed curves. They are used to collect information from the computational routines so that it can be stored in the database. In particular, `eds`, `lpds`, and `hds` contain global information specific to the continuation of equilibria, limit points, and

Hopf points, respectively. The structure `lds`, contains those for the continutation of either limit cycles or period doubling bifurcation points of limit cycles. These global variables also allow computational routines to share information.

The database of Computed Objects and Curves and the database of System Descriptions are in the directory Systems. This directory contains a MATLAB mat-file *session.mat* in which the GUI information is stored as it was left when exiting MATCONT in the previous session. When restarting MATCONT, this file is loaded and the exit situation of the previous session is restored. The file *session.mat* contains the global variable `sys` of `cont` that is discussed in Section 5.

Information on computed objects and curves is also stored by MATCONT in the directory Systems. For each studied system, say *adapt2*, the directory Systems contains an m-file *adapt2.m* that contains the definition of the system, a mat-file *adapt2.mat* in which the structure `gds` of the previous run of MATCONT with the system *adapt2* is stored and a directory `adapt2` with information on the curves is computed for the system *adapt2*.

To understand the nomenclature, we note that MATCONT distinguishes several types of objects, among which (the list is growing) are the following:

— Point: label P,
— Equilibrium: label EP,
— Limit Point: label LP,
— Hopf Point: label H,
— Bogdanov-Takens Point: label BT,
— Zero-Hopf Point: label ZH,
— Double Hopf Point: label DH,
— Cusp Point: label CP,
— Generalized Hopf Point: label GH,
— Limit Cycle: label LC,
— Period Doubling Bifurcation of Limit Cycles: label PD,
— Limit Point of Cycles: label LPC, and
— Torus Bifurcation of Cycles: label NS.

MATCONT also distinguishes several types of curves. The list contains at present: Orbits (label: O), Equilibrium, Limit Point, Hopf Point, Limit Cycle, Period Doubling Bifurcation Point of Limit Cycles and Limit Point of Cycles. All these labels have the same meaning as in CONTENT (see, e.g., Kuznetsov [1998] for a theoretical background).

The curve type Orbit is special because it is computed by an Integrator routine while all other curves are computed by the Continuer.

In the subdirectory `adapt2` of Systems, the curves computed for the system *adapt2* are stored as mat-files. This first includes the string variables `point` and `ctype` that denote, respectively, the type of the starting point of the curve and the type of the curve. For example, the combination H-LC indicates that the curve is a curve of limit cycles which was started from a Hopf point. These labels are also used to name the file of the computed curve. For example, the first curve of the system *adapt2* with labels H-LC is stored in the file H-LC(1)

the subdirectory `adapt2` of Systems. These names also appear in the SelectPoint window, where the user has the option of renaming the curves.

Also, for each computed curve the arrays `x,v` and the array of structures `s` are stored. Here `x` is simply the list of computed points, each column corresponding to a point. Similarly, `v` is the list of tangent vectors in the same points. `s` is a one-dimensional array, each element of which is a structure that corresponds to a special point on the curve. This structure has the following fields :

— `s.index`: index of the singularity point in `x` and `v`.
— `s.label`: label of the singularity point, for example, LP, H.
— `s.msg`: a string that contains a message for this type of singularity.
— `s.data`: any kind of information that is useful in the further processing of this type of singularity. For example, for Hopf points the critical eigenvalues, the corresponding eigenvectors or the first Lyapunov coefficient might be stored here.

Next, `cds` and the curve-type related global structures such as `eds`, `lds`,..., are stored. This allows the recovery of the values of the fixed parameters, the location (in the parameter vector) of the free parameters, and the computational settings and constants of the run.

4. THE GUI

The GUI is partially developed with GUIDE (the standard tool within MATLAB). This tool can only be used for windows that remain fixed. For example, the Continuer window was generated with GUIDE but the Starter window was created manually because its layout depends on the type of the curve.

To illustrate the working of the GUI, we consider the question: How does the GUI know what type of point the user has selected? Every object in the windows of the GUI has a tag. If one selects Point and then Hopf in the main MATCONT window, the following call will be made:

`matcont('point_callback')`(the function `'point_callback'` in the *matcont.m* masterfile will be executed)—*'point_callback'* contains the following code:

```
global gds;
tag=get(gcbo,'Tag');[point,str]=strtok(tag,'_');
if ~isempty(str)
    if ~isempty(gds.type)
        type=strcat('_',deblank(gds.type));
    else
        type='';
    end
    %does the current type appear in the list of possible types?
    if isempty(findstr(str,type))
        type=strtok(str,'_');%the default type is chosen
    else
```

Table I. Types of Point and Their Corresponding Tags Within the GUI

Types of point	Tag
Point	P_O_EP_LP_H
Equilibrium	EP_EP
Limit cycle	LC_LC
Period Doubling	PD_LC_PD
Limitpoint of Cycles	LPC_LC_LPC
Neimark-Sacker	NS_LC
Hopf	H_LC_EP_H
Limitpoint	LP_LP_EP
Branching Point	BP_EP
Cusp	CP_LP
Bogdanov-Takens	BT_LP
Generalized Hopf	GH_H_LC
Zero-Hopf	ZH_H_LP
Double Hopf	DH_H

```
        type=strtok(type,'_');%the current curve type is chosen
    end
    feval(type,'point',tag);
else
str=sprintf('It isn''t provided to start a continuation
from a %s point',point);
errordlg(str);
end
```

In this code `gcbo` returns the handle of the object whose callback is currently executing. The command `tag=get(gcbo,'Tag')` gets the tag corresponding to Hopf in Table I, namely, H_LC_EP_H. This tag has the following meaning:

—H is the type of point (first H in the list).

—LC is the standard type of curve. The standard type will be chosen if the current type of curve does not appear in the list of possible types of curve.

—The defaults LC, H, and EP are the possible types of curve.

Once the curve type is selected, the call `feval(type,'point',tag)` is executed. For every curve type, say, LC, there is a corresponding file *LC.m* that contains information for the GUI. In *LC.m* the part `if strcmp(arg,'point')` will be executed. This part selects the curve types allowed from the point type. It also calls `LC('type')`. That part chooses the new curve name and loads the Main window. It also loads the Continuer and Starter windows.

We now address the question: How does the GUI know which initializer to use when the user starts a computation? If one selects "compute" and then "forward" in the Main window of MATCONT, the call

```
matcont('forward_callback')
```

is executed. This routine sets some options and makes a call to `matcont('start_cont')` which in turn calls `feval(deblank(gds.type), 'start_cont')`. In our example this means that there will be call to the file *LC.m* and that the part

```
if strmp (arg,'start_cont')
```

will be executed. This part selects the appropriate initializer and starts the Continuer. Thus, the following commands will be executed:

```
[x0,v0]=init_H_LC(gds.system,x0,par,gds.options.ActiveParams,
 gds.amplitude,gds.discretization.ntst,gds.discretization.ncol);
[x,v,s,h,f]=cont('limitcycle',x0,v0,gds.options);
```

5. THE COMPUTATIONAL ROUTINES

MATLAB provides an excellent suite of ODE Integrators [Shampine and Reichelt 1997]. To provide a combined integration-continuation environment, we have made them all accessible in MATCONT. The Integrator window is opened automatically when the curve-type Orbits is selected and offers a default choice of Integrator and parameters. The user has the option to make any desired change. The orbit is plotted in real time if an appropriate two-dimensional (2D) or three-dimensional (3D) window is opened.

We note that time simulation is not only important in its own right; in a continuation environment, it can also be used to compute stable equilibria or stable limit cycles from which the continuation can start to detect and compute more complex phenomena.

Notice that the standard MATLAB routines `odeset` and `odeget` only support Jacobian matrices coded symbolically in the ODE file. This is far too restrictive for our purposes. We therefore replaced them by compatible routines (with the same names) that support derivatives with respect to parameters as well as higher-order derivatives with respect to the state variables and parameters.

The continuation kernel implementing the algorithm described in Section 2 is contained in the file *cont.m* in the directory Continuer. The global variables are `cds` and `sys`. `sys` is a structure with two fields, namely, sys.gui and `sys.file`. `sys.gui` is a structure with five fields. `sys.gui.pauseeachpoint`, `sys.gui.pausenever` and `sys.gui.pausespecial` are doubles that take on the values 0 and 1, only. Two of them are 0, the remaining one is 1, and this governs the stop-mode during the continuation of curves. `sys.gui.ncurves` corresponds to the number of stored curves of particular type, and `sys.gui.plot_points` is the number of points that have to be calculated before they are plotted. Next, `sys.file` is a character string of the form example.mat, where "example" is the name of the last system loaded by MATCONT.

For a given curve type, a number n_s of possible bifurcations is signaled and labeled in the Description of Curve Type and a number n_t of test functions is provided. Detection and computation of singularities are based on a Singularity Matrix, S which is an $(n_s \times n_t)$-matrix also given in the Description of Curve Type.

A singularity is not always fully characterized by the vanish at a single test function. It may require that other test functions also vanish or, on the contrary, do not vanish. The convention is as follows. If $S(i, j) = 0$, then the jth test function must vanish in a bifurcation of type i. If $S(i, j) = 1$, then it

must not vanish. If $S(i, j)$ is not 0 or 1, then the test function is not relevant to the bifurcation. An example is given in Section 6.

For every single run, the user can choose the bifurcation to be detected on the computed curve; these are the *active bifurcations*. Using this information and the Singularity Matrix, the Continuer makes a list of test functions that are to be monitored; these are the *active test functions*.

A vector of values of active test functions is computed at every computed point along the curve and compared with the vector in the previous point. If there are no sign changes, then no further action is taken. If there are sign changes, then the active bifurcations are considered consecutively. Suppose that j is the index of an active bifurcation. If there is a test function t_i for which $S(i, j) = 0$ and t_i does not change sign, then there is no further action with regard to bifurcation j. In the opposite case, the next step is to locate the zeros of all test functions t_i for which $S(i, j) = 0$ by a bisection method. If this process fails to converge for at least one test function, then an error message is printed and no further action is taken. In the opposite case, the distance between the zeros is considered. If it is larger than a certain threshold, then no further action is taken. If the distance is smaller than the threshold, then the located points are merged by simply taking arithmetic means of their coordinate values. If the absolute value of one of the test functions t_i for which $S(i, j) = 1$ is larger than a given threshold, then no further action is taken. In the remaining case, the bifurcation is declared to have been detected and the located point is stored, appropriately labeled, and considered as a "special point" on the curve.

We note that there are exceptional cases where the location of bifurcation points by the above standard method may be problematic. This is mainly the case if the bifurcation itself is nongeneric and appears because the dynamical system has a special structure or symmetries. A typical example is the appearance of a branching point on a curve of equilibria in the presence of symmetry. To overcome this difficulty, a special "locator" algorithm can be provided in the Description of Curve Type.

The computation of curves of Limit Points and Hopf points is based on the bordered Jacobian and bordered squared Jacobian methods, respectively, as described in Govaerts et al. [1998] and implemented in CONTENT.

For the computation of limit cycles, a discretization based on orthogonal collocation [Ascher et al. 1979; De Boor and Swartz 1973; Russell and Christiansen 1978] is used, essentially the same as in AUTO and CONTENT. The algorithms for the continuation of period doubling and fold bifurcation points of limit cycles are new; they have been analyzed in detail in [Doedel et al. 2003] but have never before been implemented in any standard software.

6. DESCRIPTION OF CURVE TYPES

To show the flexibility of MATCONT, we give some details of the implementation LimitPoint curve. This involves several tasks:

— Introduce a directory LimitPoint and set the path, that is, add a line `addpath([cd '/LimitPoint/']);` to the file *matcont.m*.

— Introduce a structure `lpds` that will be global in *cont.m* and the computational routines in the LimitPoint directory.
— Write the initializers that allow to start the curve from the appropriate starting points; in this case only a LimitPoint fits the description. The corresponding file is called *init_LP_LP.m*.
— Write the computational routine, a MATLAB m-file named *limitpoint.m* that defines Limit Points.
— Make a file *LP.m* to handle the new type of curves in MATCONT. *LP.m* contains the information about the appearance of the LP-related windows in the GUI. We note that, for example, the starter windows for the LP and LC curves are different.

We now go into detail. Since the bordering vectors are essential in the definition of the LP points, they are included in `lpds`. The fields `lpds.borders.v`, `lpds.borders.w` contain approximations v_{bor}, w_{bor} to the right and left singular vectors of the Jacobian matrix of the system in a LP point, respectively. They are kept constant during the computation of a continuation point and during the processing of special points; however, they can be adapted between such tasks. The main task of *init_LP_LP.m* is, in fact, to initialize these two vectors.

In the file *limitpoint.m*, we have to provide the possibility to compute the defining system for Limit Points by the call `limitpoint(X)` where X is a vector of doubles (which contains the state variables of (1) and two free parameters). Now an LP point is characterized by the fact that it is an equilibrium and that the scalar g, defined by solving

$$\begin{pmatrix} f_x & w_{bor} \\ v_{bor}^T & 0 \end{pmatrix} \begin{pmatrix} v \\ g \end{pmatrix} = \begin{pmatrix} 0_n \\ 1 \end{pmatrix}, \tag{8}$$

is zero (v is an n-vector). The output of a call to `limitpoint(X)` is a vector with $n + 1$ components, namely:

$$\begin{pmatrix} f(x, \alpha) \\ g \end{pmatrix}.$$

In the file *limitpoint.m*, we also have to provide the possibility to compute the derivatives of the defining system for Limit Points with respect to the state variables and active parameters by the call `limitpoint('jacobian',X)`. Now the derivatives of the first n components, that is, the equilibrium equations, are simply the Jacobian values. The derivatives of g are given by

$$g_z = -w^T (f_x)_z v,$$

where z is a state variable or an active parameter and w is obtained by solving

$$\begin{pmatrix} f_x^T & v_{bor} \\ w_{bor}^T & 0 \end{pmatrix} \begin{pmatrix} w \\ g \end{pmatrix} = \begin{pmatrix} 0_n \\ 1 \end{pmatrix}, \tag{9}$$

the transposed system of Equation (8) [Govaerts 2000, §4.1.2].

There is no need for the values of the Hessian of `limitpoint(x)`, so we can leave the content of `limitpoint('hessian',X)` empty.

The call `limitpoint('defaultprocessor',X)` allows one to do any desired postprocessing on the computed LP points. At present it computes all eigenvalues.

Three generic bifurcations can be encountered along this curve: BT, ZH, CP. We note that the order is fixed in the singularity matrix. The test functions are as follows:

—$\phi_1 = w^T v$,
—$\phi_2 = \det(2 f_x \odot I_n)$, and
—$\phi_3 = w^T f_{xx} vv$.

In these expressions, v, w are the vectors computed in Equations (8) and (9), respectively, and $2 f_x \odot I_n$ is the bialternate matrix product discussed in Section 1.

We note that the order of the test functions is again fixed in the singularity matrix. The values of the testfunctions are called by

```
limitpoint('testf',X,V).
```

The Singularity Matrix is provided by the call

```
limitpoint('singmat',X,V).
```

In our case,

$$S = \begin{pmatrix} 0 & 0 & 8 \\ 1 & 0 & 8 \\ 8 & 8 & 0 \end{pmatrix}.$$

Indeed, ϕ_2 vanishes in BT points as well as in ZH points; on the other hand, ϕ_1 vanishes in BT points but not in ZH points. This is sufficient to distinguish the two cases.

The call

```
limitpoint('locate',si,X1,V1,X2,V2)
```

invokes the locator algorithm for the bifurcation with index *si* in the part of the curve between (x^1, v^1) and (x^2, v^2). We note that v^1, v^2 are the tangent vectors in x^1, x^2, respectively but these may not really be provided. At present the curve-type LimitPoint does not provide any locators (the curve-type equilibrium provides a locator for branching points).

The call

```
limitpoint('adapt',X,V)
```

allows one to update the auxiliary variables used in the defining system of the computed branch. In the case of LP points, the bordering vectors `lpds.borders.v` and `lpds.borders.w` may require updating since they must at least be such that the matrices in Equations (8) and (9) are nonsingular. Updating is done by replacing `lpds.borders.v` and `lpds.borders.w` by the normalized vectors v, w computed in Equations (8) and (9), respectively.

During the computation of a curve, it is sometimes necessary to introduce variables and do additional computations that are common to all points of the curve. These can be relegated to a call of the type

```
limitpoint('init',X,V).
```

This option has to be provided only if the variable `WorkSpace` in `cds.options` is switched on. In this case a call

```
limitpoint('done',X,V)
```

must clear the workspace. Variables in the workspace must be set global.

In the present version of MATCONT, this facility is used to compute det $(2f_x \odot I_n)$ without loops. In the workspace, we construct $(\frac{n(n-1)}{2}, \frac{n(n-1)}{2})$ matrices eds.BiAlt_M1, eds.BiAlt_M2, and eds.BiAlt_M3, whose entries are integers between 1 and $n^2 + 1$.

In the main program, the Jacobian matrix is expanded to an $(n, n+1)$ matrix A by adding a zero column. Computing A(eds.BiAlt_M1) then amounts to replacing each entry of eds.BiAlt_M1 with value i by the value $A(i)$ where $A(i)$ is found by ordering the elements of A columnwise. The details of the construction are such that A(eds.BiAlt_M1) − A(eds.BiAlt_M2) + A(eds.BiAlt_M3) yields $2f_x \odot I_n$. MATLAB allows one to use sparse matrices in this construction, so we exploit the sparsity of $2f_x \odot I_n$ efficiently.

7. USER ACTIONS AND AN EXAMPLE

The user starts a MATCONT session by typing `matcont` in the MATLAB command window. This sets the path to all needed directories and opens the MATCONT Main window. Usually many other windows will also be opened since the system restarts at the point where it was last exited. See Figure 4 for a typical example. In this figure an equilibrium curve of the system *adapt2* is being computed.

One possible action is to introduce a new system or change or load an existing one in the SelectSystemsNew window, SelectSystemsEdit, or SelectSystemsLoad windows, respectively. The SelectSystemsEdit window of the system *adapt2* is presented in Figure 5. Filling the fields is straightforward since the MATLAB syntax is used. We note that it is not allowed to leave blanks when filling the SelectSystems windows.

Derivatives of $f(x, \alpha)$ are often needed. The user has three choices. The first- and second-order derivatives can be provided manually in the SelectSystemsEdit window (`jacobianp` and `hessianp` refer to derivatives with respect to the parameters). This is a feasible option for simple systems with few variables and parameters. The next option depends on whether the Symbolic Toolbox of MATLAB is available or not. If it is not installed, then the option is to introduce derivatives from a file; at present this is possible for derivatives up to third order. Such files can be generated by any symbolic package, for example, MAPLE. If the Symbolic Toolbox of MATLAB is installed, the derivatives are computed symbolically and the results are pasted in the ODE file. Finally, if no symbolic derivatives are available, then MATCONT uses finite difference approximations

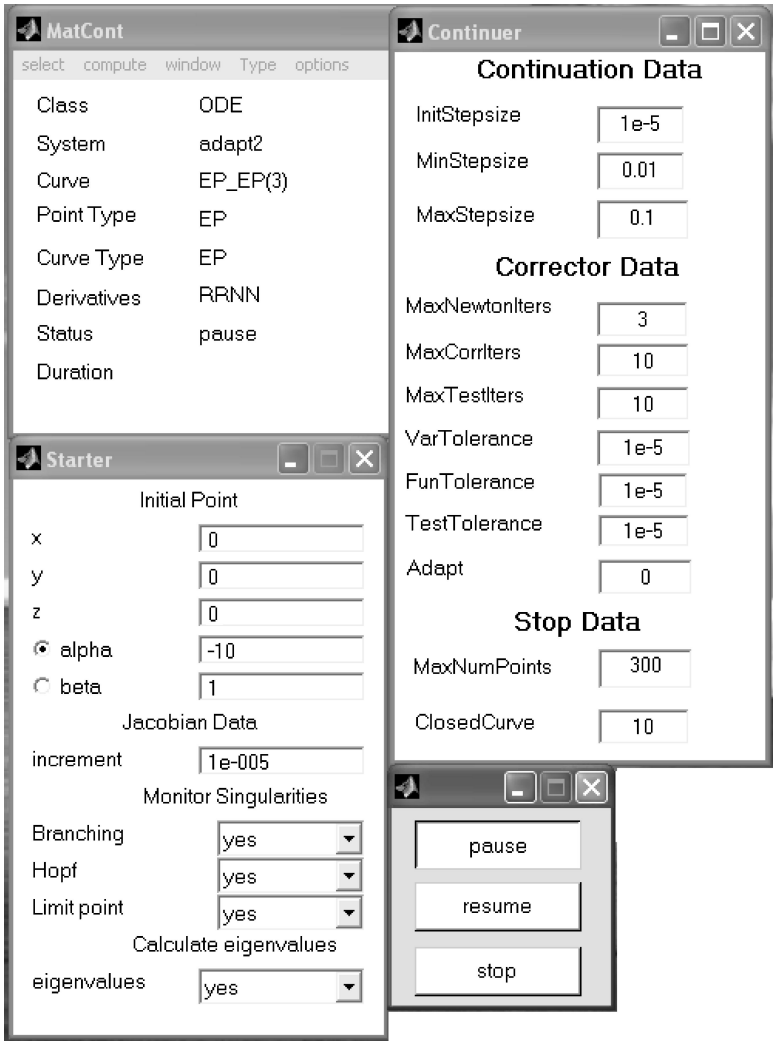

Fig. 4. Example of the Main, Continuer, EP-Starter, and Stop windows of MATCONT.

instead. In Figure 5, the first-order derivatives are input by the user while the second-order derivatives are read from file.

Another possible action from the MATCONT Main window is to select an initial point. This opens the SelectPoint window where the user gets a list of computed curves and special points found on these curves. The first and last points on each curve are also considered special. In the SelectCurve window, the user can delete or rename curves.

Suppose that in our example a curve of limit cycles is to be started from the Hopf point detected in the run presented in Figure 4. Then the user selects "initial point" from the Type menu of the MATCONT Main window. This offers a menu of curves consisting of possible types of initial points where the user chooses Hopf from the EP_EP(3) curve. The default curve to start from a Hopf

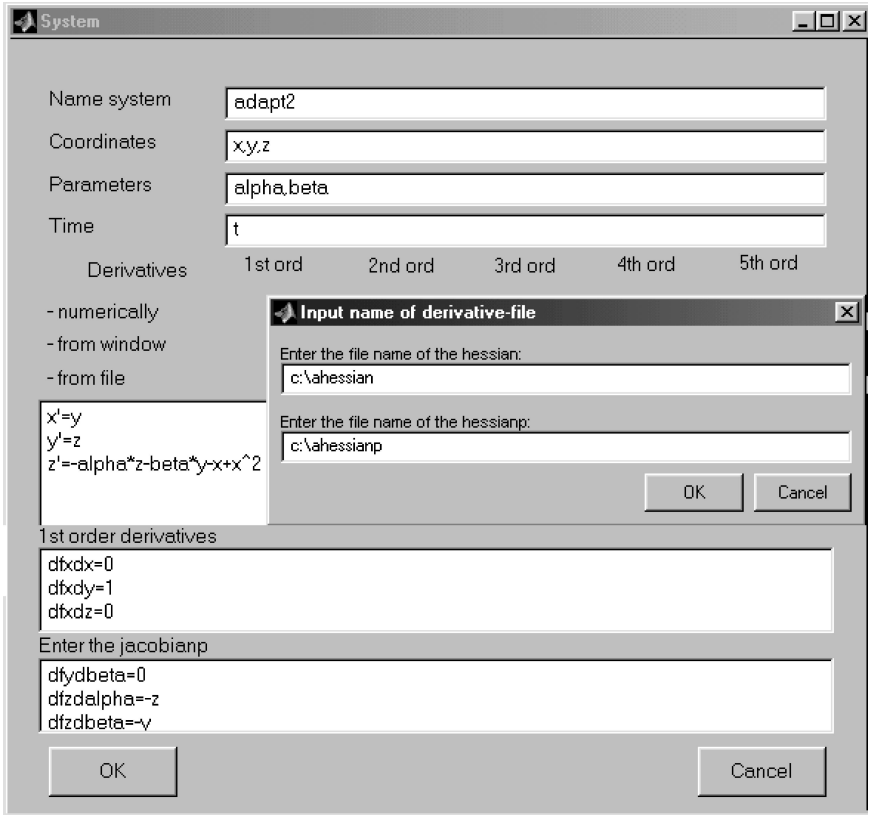

Fig. 5. Example of the SelectSystemsEdit window for the system *adapt2*.

point is an LC curve (selecting Type menu of the Main window leads to the choice between a Hopf curve and a LimitCycle curve). The Starter window and the Continuer window for limit cycles now appear. The former contains fields with the initial values for the state variables and parameters of the problem, the increment value (used in the finite difference approximations if any are needed), and the number of test intervals and collocation points for the discretization of the limit cycles. Also, there are buttons that allow choosing which bifurcations are to be detected and whether or not multipliers will be computed. The Continuer window contains parameter and threshold values for the Continuer routine and the maximum number of points to be computed. The numbers in the fields Adapt and ClosedCurve indicate after how many points the adapter routine and the check for a closed curve will be called (a zero means no call at all). All these fields are prefilled with values that are either defaults or filled in previous runs or from selecting an initial point.

The computations are started by clicking the "compute" button on the Main window. In most cases, a curve can be continued in one of two directions; therefore a further choice between forward and backward is offered. In a few cases, like starting a LimitCycle curve from a Hopf point, only one direction is meaningful and, therefore, forward and backward have the same meaning.

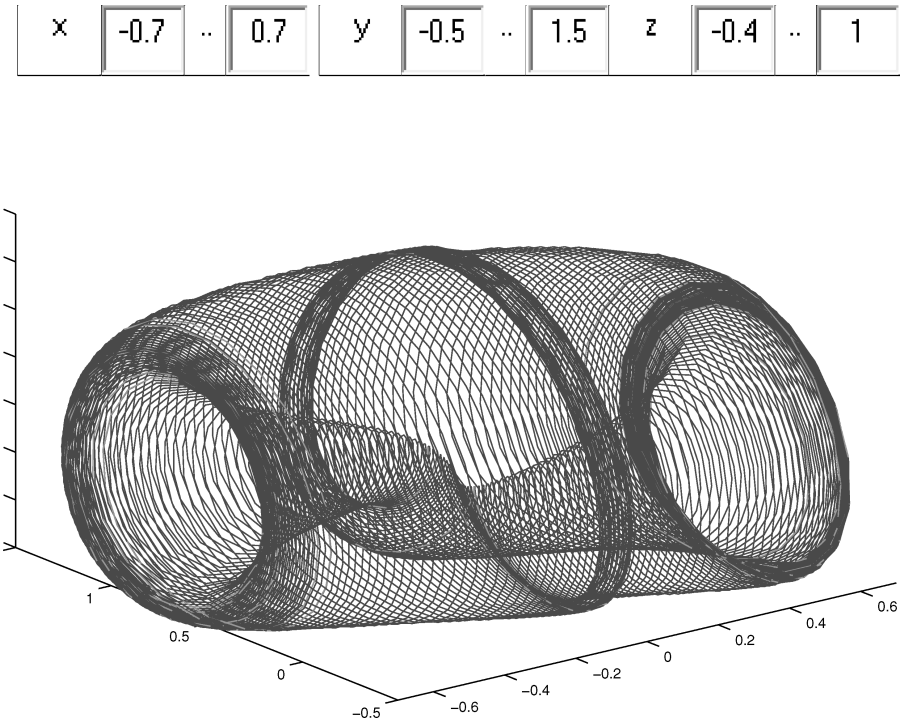

Fig. 6. Doubled period limit cycle curve started from a Period Doubling bifurcation point.

The output of the computations can be presented in real time either in numeric form or in 2D or 3D graphical form by clicking the "window" button on the Main window. In Figure 6 a 3D plot is presented. It is the result of the LimitCycle continuation started from a PD point detected in the previously described LimitCycle continuation started from a Hopf point.

The user can interact with the computations in two ways. First, the Main window has a field options that allow one to choose the pause mode. The Continuer can pause after each computed point, at special points only (the default) or not at all. Second, starting the computations automatically opens a small window in the bottom left corner of the screen with buttons "pause," "resume," and "stop" (moved to the right in Figure 2). Pressing these buttons during the computation has the corresponding effect on the ongoing computations. In Figure 4 the status is "pause" because a special point is detected. The user can stop or resume the continuation by pressing the "stop" or "resume" button, respectively. The "pause," "resume," and "stop" buttons can also be invoked by pressing a corresponding accelerator key.

APPENDIX

A. THE GDS FIELDS IN MATCONT

gds =	gds.options =
coordinates: 4x2 cell	InitStepsize: []
parameters: 9x2 cell	MinStepsize: []

```
time: 't' [0]
options: [1x1 struct]
system: 'HodgkinHuxley'
curve: [1x1 struct]
equations: [14x62 char]
dim: 4
der: [4x5 double]
jac: ''
jacp: ''
hess: ''
hessp: ''
point: 'EP '
type: 'EP '
discretization: [1x1 struct]
period: 1
plot2: [1x2 struct]
plot3: ''
open: [1x1 struct]
integrator: [1x1 struct]
numeric: [1x1 struct]
der3: ''
der4: ''
der5: ''
```

```
MaxStepsize: 1
MaxCorrIters: []
MaxNewtonIters: []
MaxTestIters: []
MoorePenrose: []
SymDerivative: []
SymDerivativeP: []
Increment: 1.0000e-005
FunTolerance: []
VarTolerance: []
TestTolerance: []
Singularities: 1
MaxNumPoints: 300
Backward: 0
CheckClosed: []
TestFunctions: []
WorkSpace: []
Locators: []
Adapt: []
IgnoreSingularity: []
ActiveParams: 2
Multipliers: 1
Eigenvalues: 1
Userfunctions: 0
UserfunctionsInfo: []
```

APPENDIX

B. THE CDS FIELDS IN MATCONT

```
cds =
curve: 'equilibrium'
ndim: 5
options: [1x1 struct]
h: 1
h_max: 1
h_min: 1.0000e-005
pJac: [4x5 double]
pJacX: [5x1 double]
h_inc_fac: 1.3000
h_dec_fac: 0.5000
nActTest: 3
S: [3x3 double]
nSing: 3
nTest: 3
ActSing: [1 2 3]
nActSing: 3
ActTest: [1 2 3]
```

```
cds.options =
InitStepsize: []
MinStepsize: []
MaxStepsize: 1
MaxCorrIters: 10
MaxNewtonIters: 3
MaxTestIters: 10
MoorePenrose: 1
SymDerivative: 0
SymDerivativeP: 0
Increment: 1.0000e-005
FunTolerance: 1.0000e-006
VarTolerance: 1.0000e-006
TestTolerance: 1.0000e-005
Singularities: 1
MaxNumPoints: 300
Backward: 0
CheckClosed: []
```

SZ: [2x4 double]	TestFunctions: []
atv: 1	WorkSpace: 1
testvals: [2x3 double]	Locators: [0 0 0]
testzero: [5x3 double]	Adapt: 0
testvzero: [5x3 double]	IgnoreSingularity: []
	ActiveParams: 2
	Multipliers: 1
	Eigenvalues: 1
	Userfunctions: 0
	UserfunctionsInfo: []

ACKNOWLEDGMENT

The authors thank Oscar De Feo (EPFL, Lausanne, Switzerland) for several helpful remarks.

REFERENCES

ALLGOWER, E. L. AND GEORG, K. 1996. Numerical path following. In *Handbook of Numerical Analysis 5*, P. G. Ciarlet and J. L. Lions, Eds. North-Holland, Amsterdam, The Neatherlands.

ARNOLD, D. AND POLKING, J. C. 1999. *Ordinary Differential Equations using* MATLAB, 2nd ed. Prentice-Hall, Englewood Cliffs, NJ.

ASCHER, U. M., CHRISTIANSEN, J., AND RUSSELL, R. D. 1979. A collocation solver for mixed order systems of boundary value problems. *Math. Comp. 33,* 146, 659–679.

BACK, A., GUCKENHEIMER, J., MYERS, M. R., WICKLIN, F. J., AND WORFOLK, P. A. 1992. DSTOOL: Computer assisted exploration of dynamical systems. *Notices Amer. Math. Soc. 39*, April, 303–309.

BEYN, W.-J., CHAMPNEYS, A., DOEDEL, E., GOVAERTS, W., KUZNETSOV, YU. A., AND SANDSTEDE, B. 2002. Numerical continuation, and computation of normal forms. In *Handbook of Dynamical Systems*, Vol. II, B. Fiedler, ed. Elsevier, Amsterdam, The Netherlands, 149–219.

CHOE, W. G. AND GUCKENHEIMER, J. 2000. Using dynamical system tools in MATLAB. In *Numerical Methods for Bifurcation Problems and Large-Scale Dynamical Systems*, IMA Vol. 119, E. J. Doedel and L. S. Tuckerman, Eds. Springer, New York, NY. 85–113.

DE BOOR, C. AND SWARTZ, B. 1973. Collocation at Gaussian points. *SIAM J. Numer. Anal. 10*, 4, 582–606.

DE FEO, O. 2000. MPLAUT: A MATLAB visualization software for AUTO97. EPFL, Lausanne, Switzerland. Available at `http://www.math.uu.nl/people/kuznet/cm`.

DHOOGE, A., GOVAERTS, W., KUZNETSOV, YU. A., MESTROM, W., AND RIET, A. 2000–2002. CL_MATCONT : A Continuation Toolbox in MATLAB. In *Proceedings of the 2003 ACM Symposium on Applied Computing* (Melbourne, FL), 161–166. Software available at: `http://www.math.uu.nl/people/kuznet/cm`.

DOEDEL, E. J., CHAMPNEYS, A. R., FAIRGRIEVE, T. F., KUZNETSOV, YU. A., SANDSTEDE, B., AND WANG, X. J. 1997. AUTO97: Continuation and bifurcation software for ordinary differential equations (with HomCont), user's guide. Concordia University, Montreal, P.Q., Canada. Available at `http://indy.cs.concordia.ca`.

DOEDEL, E., GOVAERTS, W., AND KUZNETSOV, YU. A. 2003. Computation of periodic solution bifurcations in ODEs using bordered systems. *SIAM J. Numer. Anal.* To appear.

DOEDEL, E. J., KELLER, H. B., AND KERNÉVEZ, J. P. 1991. Numerical analysis and control of bifurcation problems I : Bifurcation in finite dimensions. *Int. J. Bifurc. Chaos 1*, 3, 493–520.

ENGELBORGHS, K., LUZYANINA, T., AND ROOSE, D. 2002. Numerical bifurcation analysis of delay differential equations using DDE-BIFTOOL. *ACM Trans. Math. Softw. 28*, 1, 1–21.

ERMENTROUT, B. 2002. *Simulating, Analyzing and Animating Dynamical Systems, a Guide to XPPAUT for Researchers and Students*. SIAM Publications, Philadelphia, PA.

GOVAERTS, W., KUZNETSOV, YU. A., AND SIJNAVE, B. 1998. Implementation of Hopf and double Hopf continuation using bordering methods. *ACM Trans. Math. Softw. 24*, 4, 418–436.

GOVAERTS, W. 2000. *Numerical Methods for Bifurcations of Dynamical Equilibria*. SIAM Publications, Philadelphia, PA.

GUCKENHEIMER, J. AND HOLMES, P. 1983. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Applied Mathematical Sciences 42. Springer-Verlag, New York, NY.

HENDERSON, M. 2002. Multiple parameter continuation: Computing implicitly defined k-manifolds. *Int. J. Bifurcation Chaos 12*, 3 451–476.

KELLER, H. B. 1977. Numerical solution of bifurcation and nonlinear eigenvalue problems. *Applications of Bifurcation Theory*. Academic Press, New York, NY., 359–384.

KHIBNIK, A. I., KUZNETSOV, YU. A., LEVITIN, V. V., AND NIKOLAEV, E. V. 1993. Continuation techniques and interactive software for bifurcation analysis of ODEs and iterated maps. *Physica D 62*, 1–4, 360–371.

KUZNETSOV, YU. A. 1995/1998. *Elements of Applied Bifurcation Theory*, Applied Mathematical Sciences 112. Springer-Verlag, New York, NY.

KUZNETSOV, YU. A. AND LEVITIN, V. V. 1995–1997. CONTENT: A multiplatform environment for analyzing dynamical systems. Dynamical Systems Laboratory, CWI, Amsterdam, The Netherlands. Available at `ftp.cwi.nl/pub/CONTENT`.

MESTROM, W. 2002. Continuation of limit cycles in MATLAB. Master's thesis. Mathematical Institute, Utrecht University, Utrecht, The Netherlands.

POLKING, J. C. 1997–2003. dfield and pplane software. Available at `http://math.rice.edu/~dfield`.

RIET, A. 2000. A continuation toolbox in MATLAB. Master's thesis. Mathematical Institute, Utrecht University, Utrecht, The Netherlands.

RUSSELL, R. D. AND CHRISTIANSEN, J. 1978. Adaptive mesh selection strategies for solving boundary value problems. *SIAM J. Numer. Anal. 15,* 1, 59–80.

SHAMPINE, L. F. AND REICHELT, M. W. 1997. The MATLAB ODE suite. *SIAM J. Sci. Compt. 18,* 1, 1–22.

Received May 2002; revised January 2003; accepted February 2003