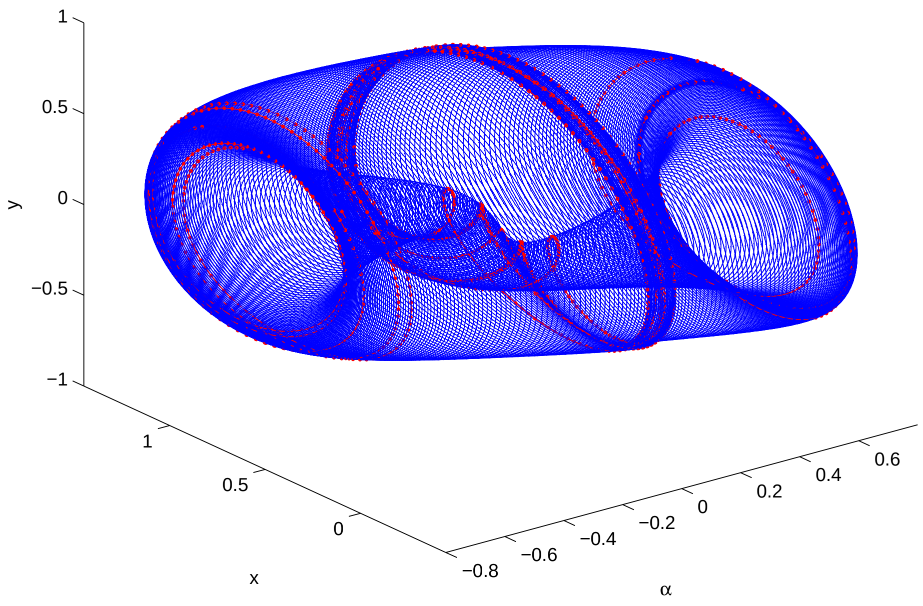
MATCONT and CL_MATCONT: Continuation toolboxes in MATLAB

W. Govaerts, Yu. A. Kuznetsov
V. De Witte, A. Dhooge, H.G.E. Meijer, W. Mestrom, A.M. Riet and B. Sautois

August 2011, adapted for version 4.1.



UNIVERSITEIT GENT Belgium

Utrecht University
The Netherlands

Contents

1 Introduction

The study of differential equations requires good and powerful mathematical software. Also, a flexible and extendible package is important. However, most of the existing software all have their own manner of specifying the system or are written in a relatively low-level programming language, so it is hard to extend it.

A powerful and widely used environment for scientific computing is MATLAB[25]. The aim of MatCont and CL_MatCont is to provide a continuation toolbox which is compatible with the standard MATLAB ODE representation of differential equations. The user can easily use his/her models without rewriting them to a specific package. The MATLAB programming language makes the use and extensions of the toolbox very easy.

This document is structured as follows. In section 2 the underlying mathematics of continuation will be treated. Section 3 introduces how singularities will be represented. The toolbox specification is explained in section 4 with a simple example in section 5. A more complex application of the toolbox, continuation of equilibria, is described in section 7. The continuation of a solution to a boundary value problem in a free parameter with the 1-D Brusselator as example is described in Section 8. Section 9 describes the continuation of limit cycles and the computation of the phase response curve. Section 10 describes the continuation of codim 1 bifurcations, at present limit points of equilibria, Hopf bifurcation points of equilibria, period doubling bifurcation points of limit cycles, fold bifurcation points of limit cycles and Neimark-Sacker bifurcation points of limit cycles. Section 11 describes the continuation of codim 2 bifurcations, at present branch points of equilibria and branch points of limit cycles. Section 12 deals with the continuation of homoclinic orbits.

1.1 Features

Upon the development of MatCont and CL_MatCont, there were multiple objectives:

- Cover as many bifurcations in ODEs as possible (e.g. now all bifurcations with two control parameters are covered).
- Allow easier data exchange between programs and with MATLAB's standard ODE solvers.
- Implement robust and efficient numerical methods for all computations, using minimally extended systems where possible.
- Represent results in a form suitable for standard control, identification, and visualization.
- Allow for easy extendibility.

A general comparison of the available features during computations for ODEs currently supported by the most widely used software packages AUTO97/2000 [9], CONTENT 1.5 [24] and MatCont/CL_MatCont are indicated in Table 1.

Relationships between objects of codimension $0, 1$ and 2 computed by MatCont and CL_MatCont are presented in Figures 1 and 2, while the symbols and their meaning are summarized in Tables 2 and 3, where the standard terminology is used, see [23].

An arrow in Figure 1 from O to EP or LC means that by starting time integration from a given point we can converge to a stable equilibrium or a stable limit cycle, respectively. In

Table 1: Supported functionalities for ODEs in AUTO (A), CONTENT (C) and MatCont (M).

	A	C	M
time-integration		+	+
Poincaré maps			+
monitoring user functions along curves computed by continuation	+	+	+
continuation of equilibria	+	+	+
detection of branch points and codim 1 bifurcations (limit and Hopf points) of equilibria	+	+	+
computation of normal forms for codim 1 bifurcations of equilibria		+	+
continuation of codim 1 bifurcations of equilibria	+	+	+
detection of codim 2 equilibrium bifurcations (cusp, Bogdanov-Takens, fold-Hopf, generalized and double Hopf)		+	+
computation of normal forms for codim 2 bifurcations of equilibria			+
continuation of codim 2 equilibrium bifurcations in three parameters		+	
continuation of limit cycles	+	+	+
computation of phase response curves and their derivatives			+
detection of branch points and codim 1 bifurcations (limit points, flip and Neimark-Sacker (torus)) of cycles	+	+	+
continuation of codim 1 bifurcations of cycles	+		+
branch switching at equilibrium and cycle bifurcations	+	+	+
continuation of branch points of equilibria and cycles			+
computation of normal forms for codim 1 bifurcations of cycles			+
detection of codim 2 bifurcations of cycles			+
continuation of orbits homoclinic to equilibria	+		+

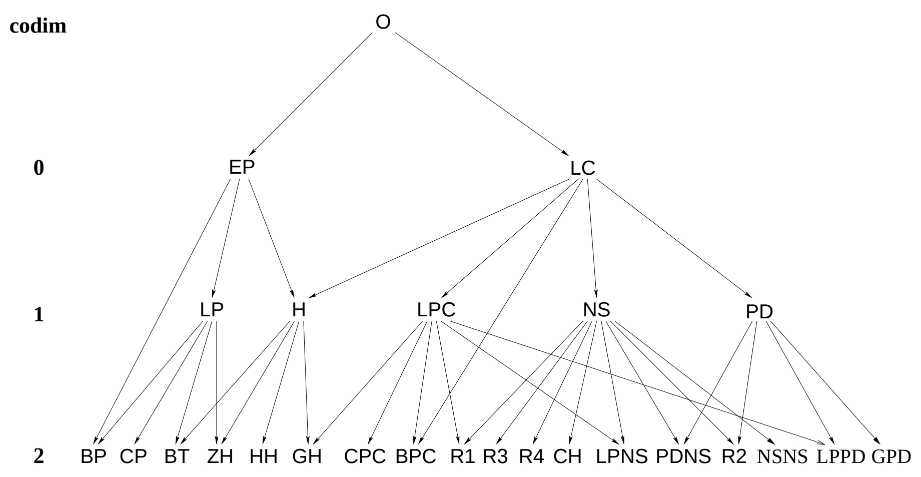

codim
0
1
2
O
EP
LC
LP
H
LPC
NS
PD
BP CP BT ZH HH GH CPC BPC R1 R3 R4 CH LPNS PDNS R2 NSNS LPPD GPD

Figure 1: The graph of adjacency for equilibrium and limit cycle bifurcations in MatCont

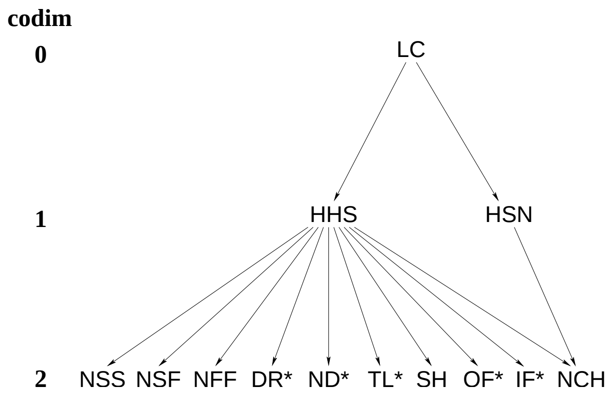


Figure 2: The graph of adjacency for homoclinic bifurcations in MatCont; here * stands for S or U.

general, an arrow from an object of type A to an object of type B means that the object of type B can be detected (either automatically or by inspecting the output) during the computation of a curve of objects of type A. For example, the arrows from EP to H, LP, and BP mean that we can detect H, LP and BP during the equilibrium continuation. Moreover, for each arrow traced in the reversed direction, i.e. from B to A, there is a possibility to start the computation of the solution of type A starting from a given object B. For example, starting from a BT point, one can initialize the continuation of both LP and H curves. Of course, each object of codim 0 and 1 can be continued in one or two system parameters, respectively.

The same interpretation applies to the arrows in Figure 2, where '*' stands for either S or U, depending on whether a stable or an unstable invariant manifold is involved.

In principle, the graphs presented in Figures 1 and 2 are connected. Indeed, it is known that curves of codim 1 homoclinic bifurcations emanate from the BT, ZH, and HH codim 2 points. The current version of MatCont fully supports, however, only one such connection: BT $\to$ HHS.

Type of object	Label
Point	P
Orbit	O
Equilibrium	EP
Limit cycle	LC
Limit Point (fold) bifurcation	LP
Hopf bifurcation	H
Limit Point bifurcation of cycles	LPC
Neimark-Sacker (torus) bifurcation	NS
Period Doubling (flip) bifurcation	PD
Branch Point	BP
Cusp bifurcation	CP
Bogdanov-Takens bifurcation	BT
Zero-Hopf bifurcation	ZH
Double Hopf bifurcation	HH
Generalized Hopf (Bautin) bifurcation	GH
Branch Point of Cycles	BPC
Cusp bifurcation of Cycles	CPC
1:1 Resonance	R1
1:2 Resonance	R2
1:3 Resonance	R3
1:4 Resonance	R4
Chenciner (generalized Neimark-Sacker) bifurcation	CH
Fold–Neimark-Sacker bifurcation	LPNS
Flip–Neimark-Sacker bifurcation	PDNS
Fold-flip	LPPD
Double Neimark-Sacker	NSNS
Generalized Period Doubling	GPD

Table 2: Equilibrium- and cycle-related objects and their labels within the GUI

1.2 Availability

This package is freely available for download at:

`http://www.sourceforge.com/`

(search for 'matcont' and then preferably go to the latest release). Unzipping the downloaded file creates a directory `matcont` or `cl_matcont` with all necessary files (see section 4.7).

1.3 Software requirements

The Continuation Toolbox requires MATLAB 6.5 or higher to be installed on your computer. No special MATLAB packages or toolboxes are necessary.

To improve performance in MatCont and CL_MatCont, there is C-code included, which will be compiled by MATLAB at the start-up of the package. In the simplest case, matlab will at the first use of MatCont ask the user which compiler he intends to use for this. The message might look somewhat like the following:

Type of object	Label
Limit cycle	LC
Homoclinic to Hyperbolic Saddle	HHS
Homoclinic to Saddle-Node	HSN
Neutral saddle	NSS
Neutral saddle-focus	NSF
Neutral Bi-Focus	NFF
Shilnikov-Hopf	SH
Double Real Stable leading eigenvalue	DRS
Double Real Unstable leading eigenvalue	DRU
Neutrally-Divergent saddle-focus (Stable)	NDS
Neutrally-Divergent saddle-focus (Unstable)	NDU
Three Leading eigenvalues (Stable)	TLS
Three Leading eigenvalues (Unstable)	TLU
Orbit-Flip with respect to the Stable manifold	OFS
Orbit-Flip with respect to the Unstable manifold	OFU
Inclination-Flip with respect to the Stable manifold	IFS
Inclination-Flip with respect to the Unstable manifold	IFU
Non-Central Homoclinic to saddle-node	NCH

Table 3: Objects related to homoclinics to equilibria and their labels within the GUI

```
Select a compiler:
[1] Lcc C version 2.4.1 in C:\PROGRAM FILES\MATLAB\R2006A\sys\lcc
[2] Microsoft Visual C/C++ version 7.1 in c:\Program Files\Microsoft...
[3] Microsoft Visual C/C++ version 6.0 in C:\Program Files\Microsoft...

[0] None

Compiler:
```

It is recommended to use the

```
Lcc C
```

-compiler. This is the built-in compiler in MATLAB, and it is with this compiler that the software is tested. So e.g. in this case, type 1 and enter. Confirmation will be asked, and after that another message may appear, which should be disregarded.

Should you want to change the compiler that is used by MATLAB, then type

```
mex -setup
```

at the command-line, and the same options will be given again.

However, compilation can depend on the Matlab version and operating system of the computer. In case of problems, check for details provided with the MatCont release

1.4 Disclaimer

The packages MatCont and CL_MatCont are freely available for non-commercial use on an “as is” basis. In no circumstances can the authors be held liable for any deficiency, fault or other mishappening with regard to the use or performance of MatCont and/or CL_MatCont.

For this manual, knowledge of dynamical systems theory is assumed.

2 Numerical continuation algorithm

Consider a smooth function $F : \mathbf{R}^{n+1} \to \mathbf{R}^n$. We want to compute a solution curve of the equation $F(x) = 0$. Numerical continuation is a technique to compute a consecutive sequence of points which approximate the desired branch. Most continuation algorithms implement a predictor-corrector method. The idea behind this method is to generate a sequence of points x_i, $i = 1, 2, \ldots$ along the curve, satisfying a chosen tolerance criterion: $||F(x_i)|| \leq \epsilon$ for some $\epsilon > 0$ and an additional accuracy condition $||\delta x_i|| \leq \epsilon'$ where $\epsilon' > 0$ and δx_i is the last Newton correction.

To show how the points are generated, suppose we have found a point x_i on the curve. Also suppose we have a normalized tangent vector v_i at x_i, i.e. $F_x(x_i)v_i = 0$, $\langle v_i, v_i \rangle = 1$.

The computation of the next point x_{i+1} consists of 2 steps:

- prediction of a new point
- correction of the predicted point

2.1 Prediction

Suppose $h > 0$ which will represent a stepsize. A commonly used predictor is *tangent prediction*:

$$X^0 = x_i + hv_i. \tag{1}$$

The choice of the stepsize is discussed in section 2.3.

2.2 Correction

We assume that X^0 is close to the curve. To find the point x_{i+1} on the curve we use a Newton-like procedure. Since the standard Newton iterations can only be applied to systems with the same number of equations as unknowns, we have to append an extra scalar condition:

$$\begin{cases} F(x) &= 0, \\ g(x) &= 0. \end{cases} \tag{2}$$

The question is how to choose the function $g(x)$.

2.2.1 Pseudo-arclength continuation

One option for choosing $g(x)$ is to select a hyperplane passing through X^0 that is orthogonal to the vector v_i:

$$g(x) = \langle x - X^0, v_i \rangle \ . \tag{3}$$

So, the Newton iteration becomes:

$$X^{k+1} = X^k - H_x^{-1}(X^k)H(X^k) \tag{4}$$

$$H(X) = \begin{pmatrix} F(X) \\ 0 \end{pmatrix}, \quad H_x(X) = \begin{pmatrix} F_x(X) \\ v_i^T \end{pmatrix} \ . \tag{5}$$

Then one can prove that the Newton iteration for (2) will converge to a point x_{i+1} on the curve from X^0 provided that the stepsize h is sufficiently small and that the curve is regular

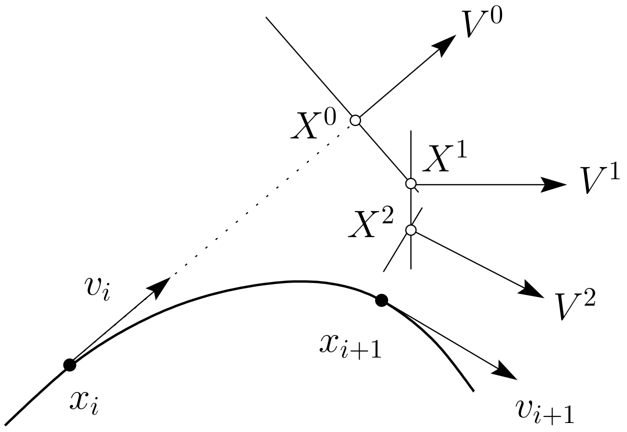


Figure 3: Moore-Penrose continuation

(rank $F_x(x) = n$). Having found the new point x_{i+1} on the curve we need to compute the tangent vector at that point:

$$F_x(x_{i+1})v_{i+1} = 0 \ . \tag{6}$$

Furthermore the direction along the curve must be preserved: $\langle v_i, v_{i+1}\rangle = 1$, so we get the $(n+1)$-dimensional appended system

$$\begin{pmatrix} F_x(x_{i+1}) \\ v_i^T \end{pmatrix} v_{i+1} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{7}$$

Upon solving this system, v_{i+1} must be normalized.

2.2.2 Moore-Penrose continuation

CL_MatCont implements a continuation method that is slightly different from the pseudo-arclength continuation.

Definition 1 *Let A be an $N \times (N+1)$ matrix with maximal rank. Then the* Moore-Penrose inverse *of A is defined by $A^+ = A^T(AA^T)^{-1}$.*

Let A be an $N \times (N+1)$ matrix with maximal rank. Consider the following linear system with $x, v \in \mathbf{R}^{N+1}, b \in \mathbf{R}^N$:

$$\begin{aligned} Ax &= b & (8)\\ v^T x &= 0 & (9) \end{aligned}$$

where x is a point on the curve and v its tangent vector with respect to A, i.e. $Av = 0$. Since $AA^+b = b$ and $v^TA^+b = \langle Av, (AA^T)^{-1}b\rangle = 0$, a solution of this system is

$$x = A^+b. \tag{10}$$

Suppose we have a predicted point X^0 using (1). We want to find the point x on the curve which is nearest to X^0, i.e. we are trying to solve the optimization problem:

$$\min_x\{||x - X^0|| \mid F(x) = 0\} \tag{11}$$

So, the system we need to solve is:

$$\begin{aligned} F(x) &= 0 && (12)\\ w^T(x - X^0) &= 0 && (13)\end{aligned}$$

where w is the tangent vector at point x. In Newton's method this system is solved using a linearization about X^0. Taylor expansion about X^0 gives:

$$\begin{aligned} F(x) &= F(X^0) + F_x(X^0)(x - X^0) + \mathcal{O}(||x - X^0||^2) && (14)\\ w^T(x - X^0) &= v^T(x - X^0) + \mathcal{O}(||x - X^0||^2)\ . && (15)\end{aligned}$$

So when we discard the higher order terms we can see using (8) and (10) that the solution of this system is:

$$x = X^0 - F_x^+(X^0)F(X^0)\ . \tag{16}$$

However, the null vector of $F_x(X^0)$ is not known, therefore we approximate it by $V^0 = v_i$, the tangent vector at x_i. Geometrically this means we are solving $F(x) = 0$ in a hyperplane perpendicular to the previous tangent vector. This is illustrated in Figure 3. In other words, the extra function $g(x)$ in (2) becomes:

$$g_k(x) = \langle x - X^k, V^k\rangle, \tag{17}$$

where $F_x(X^{k-1})V^k = 0$ for $k = 1, 2, \ldots$. Thus, the Newton iteration we are doing is:

$$\begin{aligned} X^{k+1} &= X^k - H_x^{-1}(X^k, V^k)H(X^k, V^k) && (18)\\ V^{k+1} &= V^k - H_x^{-1}(X^k, V^k)R(X^k, V^k) && (19)\end{aligned}$$

$$H(X, V) = \begin{pmatrix} F(X) \\ 0 \end{pmatrix},\quad H_x(X, V) = \begin{pmatrix} F_x(X) \\ V^T \end{pmatrix} \tag{20}$$

$$R(X, V) = \begin{pmatrix} F_x(X)V \\ 0 \end{pmatrix}\ . \tag{21}$$

One can prove that under the same conditions as for the pseudo-arclength continuation, the Newton iterations (18) and (19) converge to a point on the curve x_{i+1} and the corresponding tangent vector v_{i+1}, respectively. In the pseudo-arclength continuation, we had to compute a tangent vector when a new point was found. In this case however, we already compute the tangent vectors V^k at each iterate (19), so we only need to normalize the computed tangent vectors.

2.3 Stepsize control

Stepsize control is an important issue in these algorithms. Too small stepsizes lead to unnecessary work being done, while too big stepsizes can lead to losing details of the curve. An easily implementable and proven to be reliable method is convergence-dependent control.

Consider the computation of a next point using step size h_i. If the computation converged, let n denote the number of Newton iterations needed. Then the new step size h_{i+1} will be selected as follows:

$$h_{i+1} = \begin{cases} h_i \cdot h_{dec} & \text{if not converged} \\ h_i \cdot h_{inc} & \text{if converged and } n < n_{thr} \\ h_i & \text{otherwise} \end{cases} \tag{22}$$

where $h_{dec} < 1$, $h_{inc} > 1$ and n_{thr} are constants which are experimentally determined.

3 Singularity handling

This section explains the idea of singularities which can occur on a solution branch.

3.1 Test functions

The idea to detect singularities is to define smooth scalar functions which have regular zeros at the singularity points. These functions are called *test functions.* Suppose we have a singularity S which is detectable by a test function $\phi : \mathbf{R}^{n+1} \to \mathbf{R}$. Also assume we have found two consecutive points x_i and x_{i+1} on the curve

$$F(x) = 0, \quad F : \mathbf{R}^{n+1} \to \mathbf{R}^n \ . \tag{23}$$

The singularity S will then be detected if

$$\phi(x_i)\phi(x_{i+1}) < 0 \ . \tag{24}$$

Having found two points x_i and x_{i+1} one may want to locate the point x^* where $\phi(x)$ vanishes. A logical solution is to solve the following system

$$F(x) = 0 \tag{25}$$
$$\phi(x) = 0 \tag{26}$$

using Newton iterations starting at x_i. However, to use this method, one should be able to compute the derivatives of $\phi(x)$ with respect to x, which is not always easy. To avoid this difficulty we implemented by default a one-dimensional secant method to locate $\phi(x) = 0$ along the curve. Notice that this involves Newton corrections at each intermediate point.

3.2 Multiple test functions

The above is a general way to detect and locate singularities depending on one test function. However, it may happen that it is not possible to represent a singularity with only one test function.

Suppose we have a singularity S which depends on n_t test functions. Also assume we have found two consecutive points x_i and x_{i+1} and all n_t test functions change sign:

$$\forall j \in [1, n_t] : \phi_j(x_i)\phi_j(x_{i+1}) < 0 \tag{27}$$

Also assume we have found, using a one-dimensional secant method, all zeros x_j^* of the test functions. In the ideal (exact) case all these zeros will coincide:

$$\forall j \in [1, n_t] : x^* = x_j^* \quad \text{and} \quad \phi_j(x_j^*) = 0 \tag{28}$$

Since the continuation is not exact but numerical, we cannot assume this. However, the locations of x_j^* probably will be clustered around some center point x^c. In this case we will *glue* the points x_j^* to $x^* = x^c$.

A cluster will be detected if $\forall i, j \in [1, n_t] : ||x_i^* - x_j^*|| \leq \epsilon$ for some small value ϵ. In this case we define x^* as the mean of all located zeroes:

$$x^* = \frac{1}{n_t} \sum_{j=1}^{n_t} x_j^* \tag{29}$$

3.3 Singularity matrix

Until now we have discussed singularities depending only on test functions which vanish. Suppose we have two singularities S_1 and S_2, depending respectively on test functions ϕ_1 and ϕ_2. Namely, assume that ϕ_1 vanishes at both S_1 and S_2, while ϕ_2 vanishes at only S_2. Therefore we need a possibility to represent singularities using non-vanishing test functions.

To represent all singularities we will introduce a *singularity matrix* (as in [24]). This matrix is a compact way to describe the relation between the singularities and all test functions.

Suppose we are interested in n_s singularities and n_t test functions which are needed to detect and locate the singularities. Then let S be the $n_s \times n_t$ matrix, such that:

$$S_{ij} = \begin{cases} 0 & \text{singularity } i\text{: test function } j \text{ must vanish,} \\ 1 & \text{singularity } i\text{: test function } j \text{ must not vanish,} \\ \text{otherwise} & \text{singularity } i\text{: ignore test function } j. \end{cases} \tag{30}$$

3.4 User location

In some cases the default location algorithm can have problems to locate a bifurcation point. Therefore we provide the possibility to define a specific location algorithm for a particular bifurcation. E.g. for the localization of branch points a special algorithm is used.

4 Software

4.1 System definition

The user defines his dynamical system in a systemfile, using the framework as in the file `standard.m` in the Systems subdirectory.

In the function `fun_eval`, the dynamical system is to be given, where the parameters should be listed individually. Under `init`, the user can define some initialization parameters, as the phase variable values, the timespan, etc. All phase variables and parameters are expected to be scalar variables, not vectors or matrices. In the further functions, it is possible to supply the symbolic derivatives of the system to various orders to increase the speed and/or improve accuracy of the algorithm. Note that for the hessians or higher order derivatives, no mixed derivatives are supplied through the system definition file, since they are never used in (CL_)MatCont. The option SymDerivative indicates to which order the derivatives are provided.

Finally, it can contain the description of any number of user functions, i.e. functions that can be monitored along computed curves and whose zeros can be detected and located.

The system m-file can either be written by the user or generated automatically by the GUI of MatCont. The latter is strongly encouraged if the matlab symbolic toolbox is available and symbolic derivatives are desirable. But when using the GUI input window, users should avoid using loops, conditional statements, or other specific programming constructions in the system definition. If needed, they can be added manually to the system m-file, after MatCont has generated it.

4.2 Continuation and output

The syntax of the continuer is:

```
[x,v,s,h,f] = cont(@curve, x0, v0, options);
```

`curve` is a matlab m-file where the problem is specified (cf. section 4.3). Evaluating a function by means of a function handle replaces the earlier matlab mechanism of evaluating a function through a string containing the function name.
`x0` and `v0` are respectively the initial point and the tangent vector at the initial point where the continuation starts.
`options` is a structure as described in section 4.4.
The arguments `v0` and `options` can be omitted. In this case the tangent vector at `x0` is computed internally and default options are used.

The function returns a series of matrices:

`x` and `v` are the points and their tangent vectors along the curve. Each column in `x` and `v` corresponds to a point on the curve.

`s` is an array with structures containing information about the found singularities. Its first and last elements always refer to the first and last points of the curve, respectively, since for convenience these are also considered ”special”.

Each element of this structure array `s` has the following fields:

`s.index`	index of the singularity point in `x`, so `s(1).index` is always equal to 1 and `s(end).index` is the number of computed points.
`s.label`	label of the singularity; by convention `s(1).label` is "00" and `s(end).label` is "99".
`s.msg`	a string that may contain any information which is useful for the user, for example the full name of the detected special point.
`s.data`	For each special point it contains fields with additional information, depending on the type of point, and accumulating with increasing codimension:

- Equilibrium: s.data.v = tangent vector at the bifurcation
  - Hopf point: s.data.lyapunov = first Lyapunov coefficient
  - Limit point: s.data.a = normal form coefficient
    - Bogdanov-Takens / Zero Hopf / Double Hopf / Generalized Hopf / Cusp : s.data.c = normal form coefficient
- Limit cycle:

  s.data.multipliers = multipliers at the bifurcation

  s.data.timemesh = time mesh of the orbit at the bifurcation

  s.data.ntst = number of test intervals

  s.data.ncol = number of collocation points

  s.data.parametervalues = parameter values at the bifurcation

  s.data.T = period of the orbit at the bifurcation

  s.data.phi = bordering vector for locating PD bifurcations
  - Period-doubling point: s.data.pdcoefficient = normal form coefficient
  - Limit point of cycles: s.data.lpccoefficient = normal form coefficient
  - Neimark-Sacker point: s.data.nscoefficient = normal form coefficient
- Homoclinic to hyperbolic saddle / Homoclinic to saddle-node:

  s.data.timemesh = time mesh of the orbit at the bifurcation

  s.data.ntst = number of test intervals

  s.data.ncol = number of collocation points

  s.data.parametervalues = parameter values at the bifurcation

  s.data.T = period of the orbit at the bifurcation

`h` contains some information on the continuation process. Its columns are related to the computed points, and have the following components:

Stepsize	Stepsize used to calculate this point (zero for initial point and singular points)
Half the number of correction iterations, rounded up to the next integer	For singular points this is the number of locator iterations
User function values	The values of all active user functions
Test function values	The values of all active test functions

`f` contains different information, depending on the continuation run. For noncycle-related continuations, the `f`-vector just contains the eigenvalues, if asked for. For limit cycle continuations, it begins with the mesh points of the time- discretization, followed by, if they were asked for, the PRC- and dPRC-values in all points of the periodic orbit (cf. section 9.5). Then, if required, follow the multipliers.

It is also possible to extend the most recently computed curve with the same options (also same number of points) as it was first computed. The syntax to extend this curve is:

```
[x, v, s, h, f] = cont( x, v, s, h, f, cds);
```

`x, v, s, h` and `f` are the results of the previous call to the continuer and `cds` is the global variable that contains the curve description of the most recently computed curve (note that this variable has to be defined as `global cds` in the calling command). The function returns the same output as before, extended with the new results.

In MatCont, all curves that have been computed using a specific system are stored in separate `.mat`-files, in a directory called *diagram*, under a subdirectory named after the system. For example, curves of the *Connor* system will be kept in `.mat`-files under the subdirectory `Systems/Connor/diagram/`. For continuation runs, each such `mat`-file contains the computed `x,v,s,h,f` arrays, plus the `cds` structure and a structure related to the curve type. Also, it contains the variables *point*, *ctype* and *num*. To understand their meaning, suppose that we are computing curves of limit cycles that we start from Hopf points. The first such computed curve then gets the name "H_LC(1)", *point* stores the string "H" and *ctype* stores the string "LC". Furthermore, *num* stores the index in `s` of the last selected point of the curve (the default is 1). The second curve of the same type is called "H_LC(2)" and so on. In fact, to save storage space, only a limited number of curves of a certain type is stored. This number can be set by the user and the default is 2. To save a computed curve permanently, the user must change its name.

For time integration runs, cf. §6.1 (Curve Type O) and §6.2.1 (Curve type DO), the `mat`-file contains `ctype`, `option`, `param`, `point`, `s`, `t`, `x`. Here `t` is the vector of time points and `x` is the corresponding array of computed points. `s` contains data on the first and last computed points. The meaning of `point,ctype` is similar to the case of continuation curves. Finally, `param` is the vector of parameters of the ODE (constant during time integration) and `option` is a structure that contains optional settings for time integration.

To export the computed results of a system to a different installation of MatCont one has to copy the corresponding `m`-file, the `mat`-file and the directory of the system.

These files also contain all information needed to export the computed results to the general matlab environment, so MatCont is really an open system.

MatCont also produces graphical output. 2D and 3D graphs are plotted in matlab figure windows. Such a graph can be handled as any other graph that is produced in matlab. It can be selected using the arrow-function of the matlab figure, and the line width, line style and colour can be altered. Markers can be set on the curve. It can be copy-pasted into another matlab figure. In a figure, textboxes can be inserted and axes labels can be added. Thus the user has a plethora of possibilities to combine different MatCont output graphs into one figure.

Finally, we note that users often want to introduce new systems that are modifications of existing systems, but with slightly different sets of state variables and/or parameters. The best strategy to do this in MatCont is first to edit the existing system, change its name to a new one and click "OK" to build an m-file with a different name and no associated directory of computed curves. Afterwards, one can edit the newly created system, make all desired changes and click "OK" again.

4.3 Curve file

The continuer uses a special m-file where the type of solution branch is defined. This file, further referred to as `curve.m`, contains the following sections:

- `curve_func`: contains the evaluation of the right-hand side of that type of solution branch.
- `defaultprocessor`: is called and executed after each point computed during a continuation experiment.
- `options`: sets the default setting for the options-structure for this type of solution branch (more details are given in section 4.4).
- `jacobian`: contains the evaluation of the jacobian of that type of solution branch.
- `hessians`: contains the evaluation of the hessians of that type of solution branch.
- `testf`: contains the test functions for detecting bifurcations along the branch.
- `userf`: calls the user-defined functions if there are any.
- `process`: is called when a bifurcation point is detected, to handle any necessary output messages and storage.
- `singmat`: defines the singularity matrix of the solution branch (cf. section 3.3).
- `locate`: here specific localisation functions can be defined for bifurcations (cf. section 3.4)
- `init`: handles any special initialisations needed in the parameters or workspace.
- `done`: handles any special actions needed at the end of continuing the solution branch.
- `adapt`: this is called after every n steps, where n is user-defined. It handles any adaptation of parameters, subspaces, etc.

4.4 Options

4.4.1 The options-structure

It is possible to specify various options for the continuation run. In the continuation we use the options structure which is initially created with contset:

```
options = contset;
```

will initialize the structure. The continuer stores the handle to the options in the variable `cds.options`. Options can then be set using

```
options = contset(options, optionname, optionvalue);
```

where `optionname` is an option from the following list.

InitStepsize the initial stepsize (default: 0.01)

MinStepsize the minimum stepsize to compute the next point on the curve (default: 10^{-5})

MaxStepsize the maximum stepsize (default: 0.1)

MaxCorrIters maximum number of correction iterations (default: 10)

MaxNewtonIters maximum number of Newton-Raphson iterations before switching to Newton-Chords in the corrector iterations (default: 3)

MaxTestIters maximum number of iterations to locate a zero of a testfunction (default: 10)

Increment the increment to compute first order derivatives numerically (default: 10^{-5})

FunTolerance tolerance of function values: $||F(x)|| \leq$ `FunTolerance` is the first convergence criterium of the Newton iteration (default: 10^{-6})

VarTolerance tolerance of coordinates: $||\delta x|| \leq$ `VarTolerance` is the second convergence criterium of the Newton iteration (default: 10^{-6})

TestTolerance tolerance of test functions (default: 10^{-5})

Singularities boolean indicating the presence of a singularity matrix (default: 0)

MaxNumPoints maximum number of points on the curve (default: 300)

Backward boolean indicating the direction of the continuation (sign of the initial tangent vector) v_0 (default: 0)

CheckClosed number of points indicating when to start to check if the curve is closed (0 = do not check) (default: 50)

Adapt number of points after which to call the adapt-function while computing the curve (default: 1=adapt always)

IgnoreSingularity vector containing indices of singularities which are to be ignored (default: empty)

Multipliers boolean indicating the computation of the multipliers (default: 0)

Eigenvalues boolean indicating the computation of the eigenvalues (default: 0)

Userfunctions boolean indicating the presence of user functions (default: 0)

UserfunctionsInfo is an array with structures containing information about the userfunctions. This structure has the following fields:

`.label`	label of the userfunction
`.name`	name of this particular userfunction
`.state`	boolean indicating whether the userfunction has to be evaluated or not

PRC variable indicating the computation of the phase response curve (default: empty)

dPRC variable indicating the computation of the derivative of the phase response curve (default: empty)

Input vector representing the input given to the system for the computation of the phase response curve (default: 0)

Options also contains some fields which are not set by the user but frozen or filled by calls to the curvefile, namely:

MoorePenrose boolean indicating the use of the Moore-Penrose continuation as the Newton-like corrector procedure (default: 1)

SymDerivative the highest order symbolic derivative which is present (default: 0)

SymDerivativeP the highest order symbolic derivative with respect to the free parameter(s) which is present (default: 0)

Testfunctions boolean indicating the presence of test functions (default: 0)

WorkSpace boolean indicating to initialize and clean up user variable space (default: 0)

Locators boolean vector indicating the user has provided his own locator code to locate zeroes of test functions. Otherwise the default locator will be used (default: empty)

ActiveParams vector containing indices of the active parameter(s) (default: empty)

Some more details follow here on some of the options.

4.4.2 Derivatives of the ODE-file

In the ODE-file of the object to be continued, you can provide the derivates that are needed for the continuation algorithm or other computations. The continuer has stored the handle to the derivatives in the variables `cds.curve_jacobian`,`cds.curve_hessians`.

If `cds.symjac`$= 1$, then a call to `feval(cds.curve_jacobian, x)` must return the $(n-1) \times n$ Jacobian matrix evaluated at point x.

If `cds.symhess`$= 1$, then a call to `feval(cds.curve_hessians, x)` must return a 3-dimensional $(n-1 \times n \times n)$ matrix H such that $H(i,j,k) = \frac{\partial^2 F_i(x)}{\partial x_j \partial x_k}$.

In the present implementation in most cases `cds.symhess`$= 0$, so the ODE-file does not provide second order derivatives, since they are not needed in the algorithms used.

4.4.3 Singularities and test functions

To detect singularities on the curve one must set the option *Singularities* on. Singularities are defined using the singularity matrix, as described in section 3.3. The continuer has stored the handles to the singularities, the testfunctions and the processing of the singularities respectively in the variables `cds.curve_singmat`,`cds.curve_testf` and `cds.curve_process`.

A call to `[S,L] = feval(cds.curve_singmat)` gets the singularity matrix S and a vector of 2-character strings which are abbreviations of the singularities.

A call to `feval(cds.curve_testf, ids, x, v)` then must return the evaluation of all testfunctions, whose indices are in the integer vector `ids`, at `x` (`v` is the tangent vector at `x`). As a second return argument it should return an array of all testfunction id's which could not be evaluated. If this array is not empty the stepsize will be decreased.

When a singularity is found, a call to `[failed,s] = feval(cds.curve_process,i,x,v,s)` will be made to process singularity `i` at `x`. This is the point where computations can be done, like computing normal forms, eigenvalues, etc. of the singularity. These results can then be saved in the structure `s.data` which can be reused for further analysis. Note that the first and last point of the curve are also treated as singular.

4.4.4 Locators

It may be useful to have a specific locator code for locating certain singularities (section 3.4). To use a specific locator you must set the option *Locators*. This is a vector in which the index of an element corresponds to the index of a singularity. Setting the entry to 1 means the presence of a user-defined locator. The continuer has stored the handles to the locators in the variable `cds.curve_locator` and will then make a call to `[x,v]=feval(cds.curve_locate,i,x1,v1,x2,v2)` to locate singularity i which was detected between `x1` and `x2` with their corresponding tangent vectors `v1` and `v2`. It must return the located point and the tangent vector at that point. If the locator was unable to find a point it should return `x = []`.

4.4.5 User functions

To detect userfunctions on the curve one must set the option *Userfunctions* on. The continuer has stored the handles to the userfunctions `cds.curve_userf`. First a call to `UserInfo = contget(cds.options, 'UserfunctionsInfo', [])` is made to get information on the userfunctions. A call to `feval(cds.curve_userf, UserInfo, ids, x, v)` then must return the evaluation of all userfunctions ids, whose information is in the structure `UserInfo`, at `x` (`v` is the tangent vector at `x`). As a second return argument it should return an array of all user function id's which could not be evaluated. If this array is not empty the stepsize will be decreased.

A special point on a bifurcation curve that is specified by a user function has a structure as follows:

`s.index`	index of the detected singular point defined by the user function.
`s.label`	a string that is in `UserInfo.label`, label of the singularity.
`s.msg`	a string that is set in `UserInfo.name`.
`s.data`	an empty tangent vector or values of the user functions in the singular point.

When a change of sign of a userfunction is detected, the userfunction `i` is processed at `x`. This is the point where the results (values of the userfunction) can be saved in the structure `s.data` which can be reused for further analysis.

4.4.6 Defaultprocessor

In many cases it is useful to do some general computations for every calculated point on the curve. The results of these computations can then be used by for example the test-functions. The continuer has stored the handle to the defaultprocessor in the variable `cds.curve_defaultprocessor`.

The defaultprocessor is called as
`[failed,f,s] = feval(cds.curve_defaultprocressor,x,v,s)`.
`x` and `v` are the point on the curve and it's tangent vector. The argument `s` is only supplied if the point is a singular point, in that case the defaultprocessor may also add some data to the `s.data` field. If for some reason the default processor fails it should set `failed` to 1. This will result in a reduction of the stepsize and a retry which should solve the problem. Any information that is to be preserved, should be put in `f`. `f` must be a column vector and must be of equal size for every call to the default processor.

4.4.7 Special processors

After a singular point has been detected and located a singular point data structure will be created and initialized as described in section 4.2. If there are some special data (like eigenvalues) which may be of interest for a particular singular point then a call to `[failed,s] = feval(cds.curve_process,i,x,v,s)` should store this data in the `s.data` field. Here `i` indicates which singularity was detected and `x` and `v` are the point and tangent vector where this singularity was detected.

4.4.8 Workspace

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. The continuer has stored the handle to the initialization and cleaning of the workspace in the variables `cds.curve_init` and `cds.curve_done`. These can be relegated to a call of the type

```
feval(cds.curve_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

```
feval(cds.curve_done,x,v)
```

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

4.4.9 Adaptation

It is possible to adapt the problem while generating the curve. If *Adapt* has a value, say 5, then after 5 computed points a call to `[reeval,x,v]=feval(cds.curve_adapt,x,v)` will be made where the user can program to change the system.

For some applications it is useful to change or modify the used test functions while computing the curve (like in bordering techniques). In order to preserve the correct signs of the test functions it is sometimes necessary to reevaluate the test functions after adaptation. To do this `reeval` should be one otherwise zero. The return variables `x` and `v` should be the updated `x` and `v` which may have changed because of the changes made to the system.

4.4.10 Summary

In the following table one can see what calls can be made to the problem file and which options are involved.

Syntax of call	What it should do (options involved)
`feval(cds.curve_func,x)`	return F(x)
`feval(cds.curve_options)`	return option vector
`feval(cds.curve_jacobian,x)`	return Jacobian at `x` (*SymDerivative*≥ 1)
`feval(cds.curve_hessians,x)`	return Hessians at `x` (*SymDerivative*≥ 2)
`feval(cds.curve_init,x,v)`	initialize user variable space (*WorkSpace*)
`feval(cds.curve_done)`	destroy user variable space (*WorkSpace*)
`feval(cds.curve_defaultprocessor,x,v,s)`	initialize data for testfunctions and set some general singularity data
`feval(cds.curve_testf,ids,x,v)`	return evaluation of testfunctions `ids` at `x` (*Singularities*)
`feval(cds.curve_locate,i,x1,x2,v1,v2)`	return located singularity and tangent vector(*Locators*)
`feval(cds.curve_userf,UserInfo,ids,x,v)`	return evaluation of userfunctions ids with `UserInfo` at `x` (*Userfunctions*)
`feval(cds.curve_singmat)`	return singularity matrix (*Singularities*)
`feval(cds.curve_process,i,x)`	run processor code of singularity `i` at `x`(*Singularities*)
`feval(cds.curve_adapt,x,v)`	run adaptation code of problem (*Adapt*)

4.5 Error handling

During the continuation numerical problems may arise. For example a linear system in the Newton corrections could be ill conditioned. In such cases the continuer checks for the last warning issued by MATLAB using `lastwarn()` and decreases the step size along the curve. The same mechanism is used when a test function or a user function cannot be computed.

4.6 Structure

At this point we have discussed two components of a continuation process, the continuer itself and the curve definition. In Figure 4 the complete structure is visualized. The arrows show the flow of information between the objects. As one can see, two extra components are included: the *curve initializer* and some external ODE file.

More complicated curve definitions may have the need to be initialized. Since the continuer is called only with the start point x_0 there must be some way to initialize other parameters. Calling an initializer from a GUI or command prompt solves this problem.

The standard MATLAB `odeget` and `odeset` only support Jacobian matrices coded in the ode-file. However, we do need the derivatives with respect to the parameters. It is also useful to have higher-order symbolic derivatives available.

To overcome this problem, the package contains new versions of `odeget` and `odeset` which support Jacobians with respect to parameters and higher-order derivatives. The new routines are compatible with the ones provided by MATLAB.

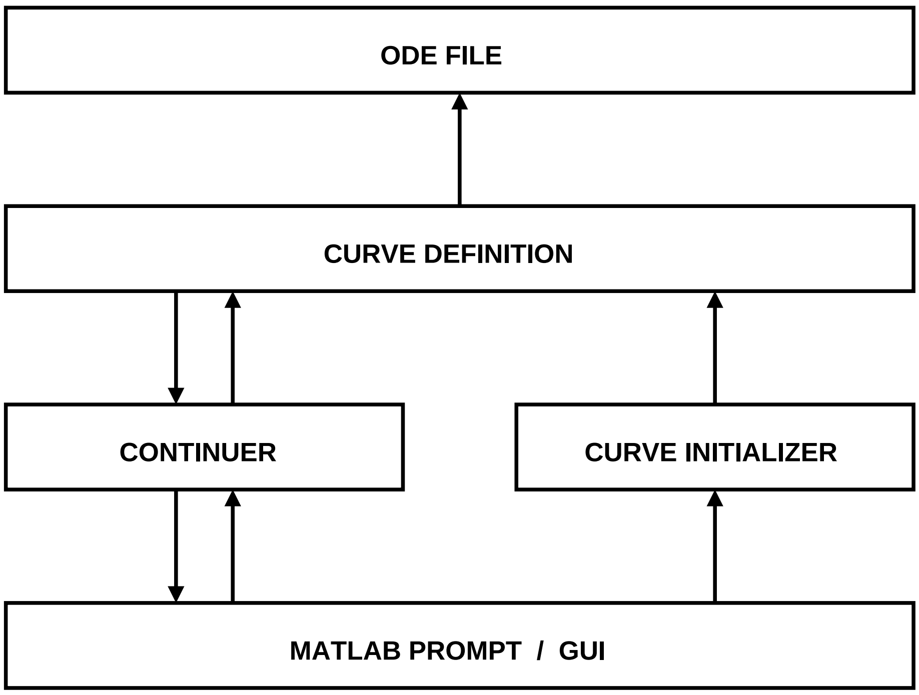


Figure 4: Structure of continuation process

To include the Jacobian with respect to parameters, the option *JacobianP* should contain the handle of the subfunction jacobianp *@jacobianp*. A call to `feval(@jacobianp, 0, x, p1, p2, ...)` should then return the Jacobian with respect to to parameter p_1, p_2, ...

To include Hessians in the ode-file the option *Hessians* should contain the handle of the subfunction hessians *@hessians*. The software then assumes that a call to `feval(@hessians, 0, x, p1, p2, ...)` will return all Hessians in the same way as mentioned above. Setting the option to *[]* indicates that there are no Hessians available from the ode-file (default behaviour).

To include Hessians with respect to parameters in your ode-file the option *HessiansP* should contain the handle of the subfunction hessiansp *@hessiansp*. The software then assumes that a call to `feval(@hessiansp, 0, x, p1, p2, ...)` will return all Hessians with respect to parameters in the same way as mentioned above. Setting the option to *[]* indicates that there are no Hessians with respect to parameters available from the ode-file (default behaviour).

To include the third order derivatives in your ode-file the option *Der3* should contain the handle of the subfunction der3 *@der3*. The software then assumes that a call to `feval(@der3, 0, x, p1, p2, ...)` will return all third order derivatives in the same way as mentioned above. Setting the option to *[]* indicates that they are not available from the ode-file (default behaviour)

Der4 and *Der5* are values indicating the 4th and 5th order symbolic derivative, available in the ode-file.

4.7 Directories

The files of the toolbox are organized in the following directories

- Continuer
  Here are all the main files stored for the continuer, which are needed to calculate and plot any curve.
- Equilibrium
  Here are all files stored needed to do an equilibrium continuation. This includes the initializers and the equilibrium curve definition file.
- LimitCycle
  Here are all files stored needed to do a limit cycle continuation. This includes the initializers and the limitcycle curve definition file.
- Limitpoint
  Here are all files stored needed to do a limitpoint continuation. This includes the initializers and the limitpoint curve definition file.
- Hopf
  Here are all files stored needed to do a Hopf point continuation. This includes the initializers and the Hopf point curve definition file.
- PeriodDoubling
  Here are all files stored needed to do a period doubling bifurcation continuation. This includes the initializers and perioddoubling curve definition files.
- LimitPointCycle
  Here are all files stored needed to do a fold bifurcation of limit cycles continuation. This includes the initializers and limitpoint of cycles curve definition files.
- NeimarkSacker
  Here are all files stored needed to do a torus bifurcation of limit cycles continuation. This includes the initializers and torus curve definition files.
- BranchPoint
  Here are all files stored needed to do a branch point continuation. This includes the initializers and a branch point curve definition file.
- BranchPointCycle
  Here are all files stored needed to do a branch point of cycles continuation. This inluded the initializers and branch point of cycles definition files.
- Homoclinic
  Here are all files stored needed to do a homoclinic-to-hyperbolic-saddle continuation. This includes the initializers and the curve definition file.
- HomoclinicSaddleNode
  Here are all files stored needed to do a homoclinic-to-saddle-node continuation. This includes the initializers and the curve definition file.

- Heteroclinc
  Here are all files stored needed to do a continuation of heteroclinic orbits. This includes the initializers and the curve definition file.
- HomtopyHet
  Here are all files stored needed to find a heteroclinic connection by the homotopy method.
- HomotopySaddle
  Here are all files stored needed to find an orbit homoclinic to saddle by the homotopy method.
- HomotopySaddlenode
  Here are all files stored needed to find an orbit homoclinic to saddle-node by the homotopy method.
- MultiLinearForms
  Here are files stored needed to compute high-order derivatives of systems and normal-form coefficients of bifurcations.
- Systems
  Here all system definition files and files related to computed curves are stored.
- GUI
  In MatCont, this directory contains all GUI-related files.
- Testruns
  Only in CL_MatCont.Here are some example testruns stored. It can be used to run the examples described in this manual and to test if everything is working correctly.
- Help
  Contains the help-files.

The only files which are not in any of these directories are in CL_MatCont `init.m` and `cpl.m` and in MatCont `matcont.fig` and `matcont.m`. The function `init` must be called before doing anything with the command-line toolbox so matlab can find all the needed functions. `cpl` is used to plot the results of the continuation in CL_MatCont. `matcont.m` is the start-up file of the GUI version MatCont, and `matcont.fig` is the related figure-file.

5 Continuer example: a circle object

In this section a simple example is presented to illustrate the basic use of the continuer. This example generates a curve g in the (x, y)-plane such that $x^2 + y^2 = 1$. So if the user specifies a point reasonably close to this curve we get the unit circle. The defining function is

$$F(x, y) = x^2 + y^2 - 1 \tag{31}$$

In the following listing this curve is implemented. Do note that while there are no options needed, the curve file *must* return an option structure (see section 4.3).

curve.m

```
function out = curve
%
% Curve file of circle
%

 out{1}  = @curve_func;
 out{2}  = @defaultprocessor;
 out{3}  = @options;
 out{4}  = []; %@jacobian;
 out{5}  = []; %@hessians;
 out{6}  = []; %@testf;
 out{7}  = []; %@userf;
 out{8}  = []; %@process;
 out{9}  = []; %@singmat;
 out{10} = []; %@locate;
 out{11} = []; %@init;
 out{12} = []; %@done;
 out{13} = @adapt;
function f = curve_func(arg)
  x = arg;
  f = x(1)^2+x(2)^2-1;

function varargout = defaultprocessor(varargin)
  if nargin > 2
    s = varargin{3};
    varargout{3} = s;
  end
  % no special data
  varargout{2} = [];
  % all done succesfully
  varargout{1} = 0;

function option = options
  option = contset;

function [res,x,v] = adapt(x,v)
  res=[];

```

curve.m

Starting computations at $(x, y) = (1, 0)$, the output in MATLAB looks like:

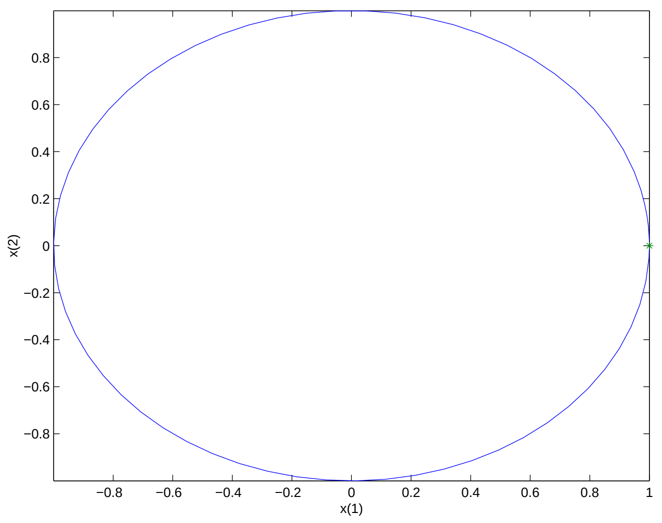


Figure 5: Computed curve of `curve.m`

```
>> init;
>> [x,v,s]=cont(@curve,[1;0]);
first point found
tangent vector to first point found
Closed curve detected at step 70

elapsed time  = 0.1 secs
npoints curve = 70
```

`cpl(x, v, s, e)` makes a two or three dimensional plot. The fourth argument `e` is optional. `x, v` and `s` are the results of the previous continuation, `e` is an array whose elements define which coordinates of the system are used. Therefore `e` must have either 2 components (2D-plot) or 3 components (3D-plot). If e is not given and `x` has 2 (respectively 3) components, then a 2D-plot (3D-plot,respectively) is drawn. In all other cases an error message will be generated. The generated curve is plotted in Figure 5 with the command:

```
>> cpl(x,v,s)
```

In this case `x` has dimension 2, so a 2D-plot is drawn with the first component of `x` (value of the state variable x) on the x-axis and the second component of `x` (value of the state variable y) on the y-axis.

6 Time integration and Poincaré maps

MATLAB provides a suite of ODE-solvers. To create a combined integration-continuation environment, we have made them all accessible in MatCont and added two new ones, `ode78` and `ode87`. `ode78` is an explicit Runge-Kutta method that uses 7th order Fehlberg formulas [Fehlberg 1969]. This is a 7-8th-order accurate integrator, therefore the local error normally expected is $O(h^9)$. `ode87` is a Runge-Kutta method that uses 8-7th order Dorman and Prince formulas. See [Prince and Dorman 1981]. This is a 8th-order accurate integrator therefore the local error normally expected is $O(h^9)$. In MatCont the choice of the solver is made via the Integrator window that is opened automatically when the curve type O (Orbit) is chosen. We note that the Starter window has a SelectCycle button that allows to start the continuation of periodic orbits from curves computed by time integration.

6.1 Time integration

MatCont uses the MATLAB representation of ODEs, which can also be used by the MATLAB ODE solvers. The user of Cl_MatCont is therefore also able to use his/her model within the standard ODE solver without rewriting the code. In MatCont, also backward time integration is possible using the standard ODE solvers. To make them accessible in MatCont, some output functions and properties were created.

6.1.1 Solver output properties

The solver output properties allow one to control the output that the solvers generate. MatCont detects whether extra output is needed by looking at the number of open output windows (2D, 3D or numeric). If extra output is required, MatCont sets the OutputFcn property to the output function, `integplot` which is passed to an ODE solver by `options = odeset('OutputFcn', @integplot)`, otherwise it is set to the function `integ_prs`. The output function must be of the form `status = integplot(t, y, flag,` p_1`,` p_2`,...)`. The solver calls this function after every successful integration step with the following flags. Note that the syntax of the call differs with the flag. The function must respond appropriately:

- `init`: The solver calls `integplot(tspan,y0,'init')` before beginning the integration, to allow the output function to initialize. This part initializes the output windows and does some initializations to speed up the further processing of the output. It also launches the window that makes it possible to interactively "stop/pause/resume" the computations.
- {`none`}: within MatCont it is possible to define the number of points (*npoints*) after which output is needed. Because the solver calls `status = integplot (t,y)` after each integration step, the number of calls to `integplot` and *npoints* does not correspond. Therefore this part is divided into two parts. `t` contains points where output was generated during the step, and `y` is the numerical solution at the points in `t`. If t is a vector, the i-th column of `y` corresponds to the i-th element of `t`. The output is produced according to the Refine option. `integplot` must return a status output value of 0 or 1. If *status* $= 1$, the solver halts integration. This part also handles the "stop/pause/resume" interactions.

- `done`: The solver calls `integplot([],[],'done')` when integration is completed to allow the output function to perform any cleanup chores. The stop-window is also deleted.

Setting the OutputFcn property to the output function, `integ_prs`, reduces the output time. It only allows to interactively pause, resume and stop the integration.

6.1.2 Jacobian matrices

Stiff ODE solvers often execute faster if the Jacobian matrix, i.e. the matrix of partial derivatives of the RHS that defines the differential equations, is provided. The Jacobian matrix pertains only to those solvers for stiff problems (`ode15s, ode23s, ode23t, ode23tb`), for which it can be critical for reliability and efficiency. If the user does not provide a code to calculate the Jacobian matrix, these solvers approximate it numerically using finite differences. Supplying an analytical Jacobian matrix often increases the speed and reliability of the solution for stiff problems. If the Jacobian matrix is provided (symbolically) in the odefile, MatCont stores as property the function handle of the Jacobian, otherwise this handle is set empty.

The standard matlab `odeget` and `odeset` only support Jacobian matrices of the RHS w.r.t. the state variables. However, we do need derivatives with respect to the parameters for the continuation. To compute normal form coefficients, it also useful to have higher-order symbolic derivatives. To overcome this problem, MatCont contains new versions of `odeget` and `odeset`, which support Jacobian matrices with respect to parameters and higher-order partial derivatives w.r.t state variables. The new routines are compatible with the ones provided by matlab.

6.2 Poincaré section and Poincaré map

A Poincaré section is a surface in phase space that cuts across the flow of a dynamical system. It is a carefully chosen (in general, curved) surface in the phase space that is crossed by almost all orbits. It is a tool developed by Poincaré for visualization of the flow in more than two dimensions. The Poincaré section has one dimension less than the phase space and the Poincaré map transforms the Poincaré section onto itself by relating two consecutive intersection points, say u_k and u_{k+1}. We note that only those intersection points count, which come from the same side of the section. The Poincaré map is invertible because one gets u_n from u_{n+1} by following the orbit backwards. A Poincaré map turns a continuous-time dynamical system into a discrete-time one. If the Poincaré section is carefully chosen no information is lost concerning the qualitative behaviour of the dynamics. For example, if the system is being attracted to a limit cycle, one observes dots converging to a fixed point in the Poincaré section.

6.2.1 Poincaré maps in MatCont

In some ODE problems the timing of specific events is important, such as the time and place at which a Poincaré section is crossed. For this purpose, MatCont provides the special curve type DO (Discrete Orbit) which is similar to O (Orbit) but provides a Starter window in which one can specify several event functions of the form $G(u) = \sum_{j=1}^{n} a_j u_j - a_0$, so $G(u) = 0$ corresponds to a Poincaré section (to avoid nonlinear interpolation problems, only this form of function G is allowed). While integrating, the ODE solvers can detect such events by locating transitions to, from or through zeros of $G(u(t))$). One does this by setting the Events property to a function handle, e.g. `@events`, and creating a function `[value,isterminal,direction] = events(t,y)` and calling

```
[t,Y,TE,YE,IE] = solver(odefun,tspan,y0,options).
```

For the i-th event function:

- `value(i)` is the value of the function.
- `isterminal(i)` $= 1$ if the integration is to terminate at a zero of this event function and 0 otherwise.
- `direction(i)` $= 0$ if all zeros are to be computed (the default), $+1$ if only the zeros are needed where the event function increases, and -1 if only the zeros where the event function decreases.

Corresponding entries in `TE`, `YE`, and `IE` return, respectively, the time at which an event occurs, the solution at the time of the event, and the index i of the event function that vanishes.

Unfortunately this method can only be used in CL_MatCont. MatCont needs some interaction between (`t,Y`) and (`TE,YE`). This is not possible by using the Events property. The part that handles the Poincaré map is the OutputFcn property of the integrators which is now set to `integplotDO`. The user is able to enter the equation $G(u)$ in the Starter window. This output function is divided into three parts:

- The solver calls `integplotDO(tspan,y0,'init')` before starting the integration, to initialize the output function. Here $G(u(t_0))$ is initialized. The same output initializations are done as in the function `integplot`(see §6.1.1).
- The solver calls `integplotDO(tspan,y0)`. We are looking for a value t^* for which $G(u(t^*)) = 0$. The function `integDOzero` calculates the value of $G(u(t_i))$ and looks for a sign change. If there is one, the function tries to locate it. Afterwards, some processing is done.
- The solver calls `integplotDO(tspan,y0,'done')` when the integration is completed, to allow the output function to perform any cleanup chores.

6.2.2 The Steinmetz-Larter example

In the peroxidase-oxidase reaction a peroxidase catalyzes an aerobic oxidation:

$$2YH_2 + O_2 + 2H^+ \rightarrow 2YH^+ + 2H_2O,$$

where YH_2 ($NADH$) is a general electron donor. Under the proper conditions this reaction yields oscillations in the concentrations of YH_2, O_2 and various reaction intermediates. Steinmetz and Larter [Steinmetz and Larter 1991] derived the following system of four coupled nonlinear differential rate equations:

$$\left\{ \begin{array}{rcl} \dot{A} & = & -k_1ABX - k_3ABY + k_7 - k_{-7}A, \\ \dot{B} & = & -k_1ABX - k_3ABY + k_8, \\ \dot{X} & = & k_1ABX - 2k_2X^2 + 2k_3ABY - k_4X + k_6, \\ \dot{Y} & = & -k_3ABY + 2k_2X^2 - k_5Y, \end{array} \right. \tag{32}$$

Variable	*Value*	*Parameter*	*Value*	*Parameter*	*Value*
A	1.8609653	k_1	0.1631021	k_5	1.104
B	25.678306	k_2	1250	k_6	0.001
X	0.010838258	k_3	0.046875	k_7	0.71643356
Y	0.094707061	k_4	20	k_8	0.5
				k_{-7}	0.1175

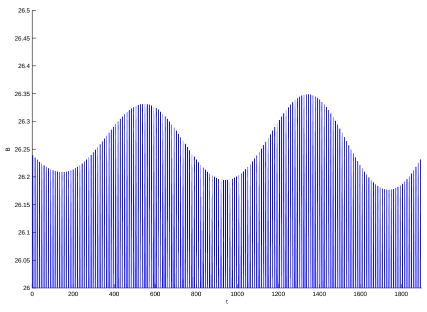
Table 5: Values of a point on the first NS- cycle in the Steinmetz-Larter model.

Variable	*Value*	*Parameter*	*Value*
A	6.1231735	k_7	1.5163129
B	9.1855407	k_8	0.83200664
X	0.0054271408		
Y	0.024602951		

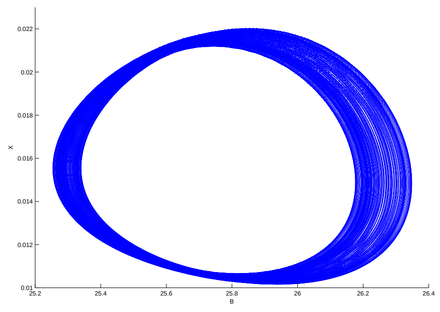
Table 6: Values of the second NS point in the Steinmetz-Larter model.

where A, B, X, Y are state variables and k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, and k_{-7} are parameters. Although highly simplified, the model is able to reproduce the three modes of simple, chaotic, and bursting oscillations found in experiments. State and parameter values of a point on a NS cycle in (32) are given in Table 5. The normal form coefficient of the NS cycle is $-1.406017e - 006$. Since it is negative, we expect stable tori nearby.

If we start a time integration from this NS point, with a slightly supercritical parameter value, namely $k_7 = 0.7167$, then after a transient the time-series exhibits modulated oscillations with two frequencies near the original limit cycle (see Fig. 6). This is a motion on a stable two-dimensional torus that arises from the Neimark-Sacker bifurcation.



(a) Modulated oscillations.



(b) Orbits on a stable 2-torus.

Figure 6:

State and parameter values of another NS point of (32) are given in Table 6. The corresponding normal form coefficient is -2.919895e-008, so that this bifurcation should also generate a stable torus. The NS point can be used as a starting point for the computation of a discrete orbit in the Poincaré section (called DO in the GUI) for a slightly decreased value of k_7, e.g. $k_7 = 1.51$. The output is shown in Figures 7 and 8, in which a cross-section of a stable torus is visible.

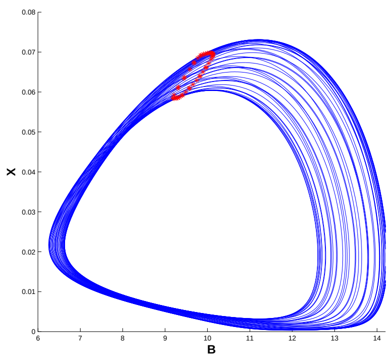


Figure 7: Dynamics on a stable torus

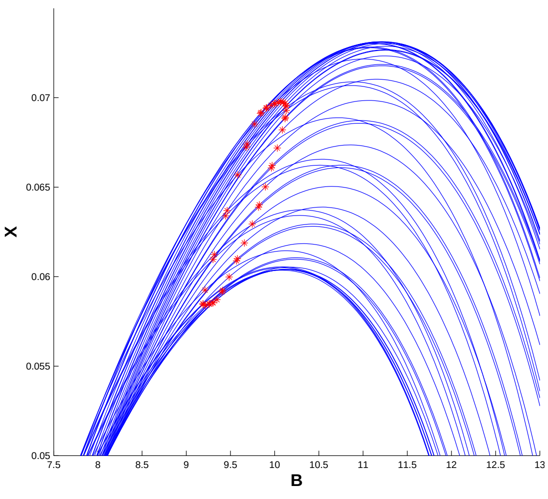


Figure 8: Poincaré map with a stable torus near the second NS bifurcation.

7 Equilibrium continuation

This example will show how to continue an equilibrium of a differential equation defined in a standard MATLAB ODE file. Furthermore, this example illustrates the detection, location, and processing of singularities along an equilibrium curve.

7.1 Mathematical definition

Consider a differential equation

$$\frac{du}{dt} = f(u,\alpha), \quad u \in \mathbf{R}^n, \alpha \in \mathbf{R} \quad f : \mathbf{R}^{n+1} \to \mathbf{R}^n \ . \tag{33}$$

We are interested in an equilibrium curve, i.e. $f(u,\alpha) = 0$. The defining function is therefore:

$$F(x) = f(u,\alpha) = 0 \tag{34}$$

with $x = (u,\alpha) \in \mathbf{R}^{n+1}$. Denote by $v \in \mathbf{R}^{n+1}$ the tangent vector to the equilibrium curve at x.

7.2 Bifurcations

In continuous-time systems there are two generic codim 1 bifurcations that can be detected along the equilibrium curve (no derivations will be done here; for more detailed information see [23]):

- *fold*, also known as *limit point*. We will denote this bifurcation by `LP`
- *Hopf*-point, denoted by `H`

The equilibrium curve can also have *branch points*. These are denoted with `BP`. To detect these singularities, we first define 3 test functions:

$$\begin{aligned}
\phi_1(u,\alpha) &= \det\begin{pmatrix} F_x \\ v^T \end{pmatrix}, & (35)\\
\phi_2(u,\alpha) &= \left(\begin{bmatrix} (2f_u(u,\alpha)\odot I_n) & w_1 \\ v_1^T & d \end{bmatrix} \setminus \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}\right)_{n+1}, & (36)\\
\phi_3(u,\alpha) &= v_{n+1}, & (37)
\end{aligned}$$

where $\odot$ is the bialternate matrix product and v_1, w_1 are $\frac{n(n-1)}{2}$ vectors chosen so that the square matrix in (36) is non-singular. Using these test functions we can define the singularities:

- `BP`: $\phi_1 = 0$
- `H`: $\phi_2 = 0$
- `LP`: $\phi_3 = 0, \ \phi_1 \neq 0$

A proof that these test functions correctly detect the mentioned singularities can be found in [23]. Here we only notice that $\phi_2 = 0$ not only at Hopf points but also at *neutral saddles*, i.e. points where f_x has two real eigenvalues with sum zero. So, the singularity matrix is:

$$S = \begin{pmatrix} 0 & - & - \\ - & 0 & - \\ 1 & - & 0 \end{pmatrix} \tag{38}$$

For each detected limit point, the corresponding *quadratic normal form coefficient* is computed:

$$a = \frac{1}{2} p^T f_{uu}[q,q], \tag{39}$$

where $f_u q = f_u^T p = 0, q^T q = 1, p^T q = 1$. At a Hopf bifurcation point, the *first Lyapunov coefficient* is computed by the formula

$$l_1 = \frac{1}{2}\mathrm{Re}\left\{p^T\left(f_{uuu}[q,q,\bar{q}] - 2f_{uu}[q, F_u^{-1} f_{uu}[q,\bar{q}]] + f_{uu}[\bar{q},(2i\omega I_n - f_u)^{-1} f_{uu}[q,q]]\right)\right\}, \tag{40}$$

where $f_u q = i\omega q,\ q^T q = 1,\ f_u^T p = -i\omega p,\ \bar{p}^T q = 1$.

7.2.1 Branch point locator

The location of Hopf and limit points usually does not cause problems. However, the location of branch points can give problems. The region of attraction of the Newton type continuation method which is used, has the shape of a cone (see [1]). In the localisation process we cannot assume to stay in this cone. This difficulty can be avoided by introducing $p \in \mathbf{R}^n$ and $\beta \in \mathbf{R}$ and considering the *extended system*:

$$\left\{\begin{array}{rcl} f(u,\alpha) + \beta p & = & 0 \\ f_u^T(u,\alpha)p & = & 0 \\ p^T f_\alpha(u,\alpha) & = & 0 \\ p^T p - 1 & = & 0 \end{array}\right. \tag{41}$$

We solve this system by Newton's method with initial data $\beta = 0$ and $p : f_u^T p = \mu p$ where μ is the real eigenvalue with smallest norm. A branch point (u,α) corresponds to a regular solution $(u,\alpha,0,p)$ of system (41) (see [3],p. 165). We note that the second order partial derivatives (Hessian) of f with respect to u and α are required.

The tangent vector at the singularity is also computed here. This is related to the processing of the branch point (computing the direction of the secondary branch).

7.3 Equilibrium initialization

Naively, one would start the continuation immediately with:

```
[x,v,s,h,f]=cont(@equilibrium, x0, v0, opt)
```

However, the equilibrium curve file has to know :

- which ode file to use,
- the values of all state variables,

- the values of all parameters,
- which parameter is active.

All this information can be supplied by one of the following two starting functions. Both functions return an initial point `x0` as well as its tangent vector `v0`.

- `[x0,v0]=init_EP_EP(@odefile, x, p, ap)`

  This routine stores its information in a global structure `eds` (see also Figure 4). The result of `init_EP_EP` is a vector `x0` with the state variables and the active parameter and a vector `v0` that is empty. Here `odefile` is the ode-file (system-definition file) to be used, `x` is a vector containing the values of the state variables. `p` is the vector containing the current values of the parameters and `ap` is the active parameter. The full listing of the equilibrium initializer can be found in the file 'Equilibrium/init_EP_EP.m' of the toolbox.

- `[x0,v0]=init_BP_EP(@odefile, x, p, s, h)`

  Calculates an initial point for starting a new branch from a branch point detected on an equilibrium curve. This routine stores its information in a global structure `eds` (see also Figure 4). Here `odefile` is the ode-file (system-definition file) to be used, `x` is a vector containing the values of the state variables returned by a previous equilibrium curve continuation. `p` is the vector containing the current values of the parameters and `h` contains the value of the initial amplitude. The full listing of the equilibrium initializer and the curve definition can be found in the files 'Equilibrium/init_BP_EP.m' and 'equilibrium.m' of the toolbox, respectively.

7.4 Bratu example

The first example we will look at is a 4-point discretization of the Bratu-Gelfand BVP [19]. This model is defined as follows:

$$\begin{aligned} x' &= y - 2x + ae^x && (42) \\ y' &= x - 2y + ae^y && (43) \end{aligned}$$

The system is specified as

bratu.m

```
function out = bratu
out{1} = @init;
out{2} = @fun_eval;
out{3} = @jacobian;
out{4} = [];
out{5} = @hessians;
out{6} = @hessiansp;
out{7} = [];
out{8} = [];
out{9} = [];
out{10}= @userf1;

end

```

```
% --------------------------------------------------------------------------
function dydt = fun_eval(t,kmrgd,a)
dydt =  [ -2*kmrgd(1)+kmrgd(2)+a*exp(kmrgd(1));
           kmrgd(1)-2*kmrgd(2)+a*exp(kmrgd(2)) ];

% --------------------------------------------------------------------------
function [tspan,y0,options] = init
tspan = [0; 10];
y0 = [0;0];handles = feval(@bratu)
options = odeset('Jacobian',handles(3),'JacobianP', 'handles(4)',...
...,'Hessians',handles(5), 'Hessiansp',handles(6));
% --------------------------------------------------------------------------
function jac = jacobian(t,kmrgd,a)
jac  = [ -2+a*exp(kmrgd(1))   1
          1              -2+a*exp(kmrgd(2)) ];

% --------------------------------------------------------------------------
function jacp = jacobianp(t,kmrgd,a)

jacp = [ exp(kmrgd(1))
         exp(kmrgd(2)) ];

% --------------------------------------------------------------------------
function hess = hessians(t,kmrgd,a)
hess1=[[a*exp(kmrgd(1)),0];[0,0]];
hess2=[[0,0];[0,a*exp(kmrgd(2))]];
hess(:,:,1) = hess1;
hess(:,:,2) = hess2;

% --------------------------------------------------------------------------
function hessp = hessiansp(t,kmrgd,a)
hessp1=[[exp(kmrgd(1)),0];[0,exp(kmrgd(2))]];
hessp(:,:,1) = hessp1;

%---------------------------------------------------------------------------
function userfun1 = userf1(t,kmrgd,a)
userfun1 = a-0.2;

```

bratu.m

As seen above, a user function is defined to detect all points where $a = 0.2$. This system has an equilibrium at $(x, y, a) = (0, 0, 0)$ which we will continue with respect to a. We first compute 50 points and then extend the curve with another 50 points.

```
>> global cds
>> p=0;ap=1;
>> [x0,v0]=init_EP_EP(@bratu,[0;0],p,ap);
>> opt=contset;opt=contset(opt,'MaxNumPoints',50);
>> opt=contset(opt,'Singularities',1);
>> opt=contset(opt,'Userfunctions',1);
>> UserInfo.name='userf1';UserInfo.state=1;UserInfo.label='u1';
>> opt=contset(opt,'UserfunctionsInfo',UserInfo);
>> [x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
```

```
first point found
tangent vector to first point found
label = u1, x = ( 0.259171 0.259171 0.200000 )
label = LP, x = ( 1.000000 1.000000 0.367879 )
a=3.535534e-001
label = H , x = ( 2.000000 2.000000 0.270671 )
Neutral saddle
label = u1, x = ( 2.542641 2.542641 0.200000 )

elapsed time  = 0.9 secs
npoints curve = 50
>> [x,v,s,h,f]=cont(x,v,s,h,f,cds);
start computing extended curve
label = BP, x = ( 3.000000 3.000000 0.149361 )

elapsed time  = 0.2 secs
npoints curve = 100
```

At $(x, y, a) \approx (3.0; 3.0; 0.15)$ the system has a branch point. To select this point the output s is used. Since the first and last points are also treated as singular, the array of structures `s` has 7 components.

```
>> cpl(x,v,s,[3 1 2]);
```

The script `cpl.m` allows to plot two- or three-dimensional curves. This routine automatically places labels at the singularity points. `cpl(x,v,s,[3 1 2])` plots a 3D-plot with the parameter a on the x-axis and the first and second state variable on the y- and z-axis. The resulting curve is plotted in Figure 9.

The labels of the plot are changed manually by the following commands:

- press 'Edit'→'Axes Properties',
- go to labels,
- enter 'a' in the Xlabel window, 'x' in the Ylabel and 'y' in the Zlabel,
- close the subwindow.

To switch to another branch at the detected branch point, we select that branch point and we use the starter `init_BP_EP`. We start a forward and backward continuation from this point.Note that in general p is a vector containing the values of all parameters at the branch point.

```
>> x1 = x(1:2,s(6).index); p(ap) = x(3,s(6).index);
>> [x0,v0]=init_BP_EP(@bratu, x1, p, s(6), 0.01);
>> opt = contset(opt,'InitStepsize',0.0001);
>> [x1,v1,s1]=cont(@equilibrium,x0,v0,opt);
first point found
tangent vector to first point found
```

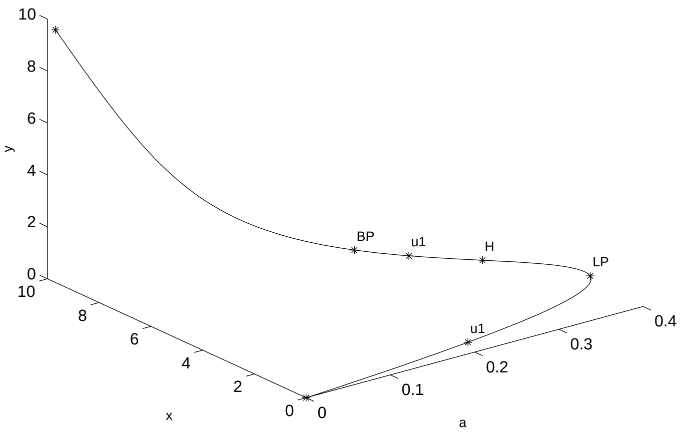


Figure 9: Equilibrium curve of `bratu.m`

```
elapsed time  = 0.5 secs
npoints curve = 50
>> cpl(x1,v1,s1,[3 1 2]);
>> opt=contset(opt,'Backward',1);
>> [x2,v2,s2]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = BP, x = ( 3.000000 3.000000 0.149361 )
elapsed time  = 0.2 secs
npoints curve = 50
>> cpl(x2,v2,s2,[3 1 2]);
```

Both curves are plotted in Figure 10.

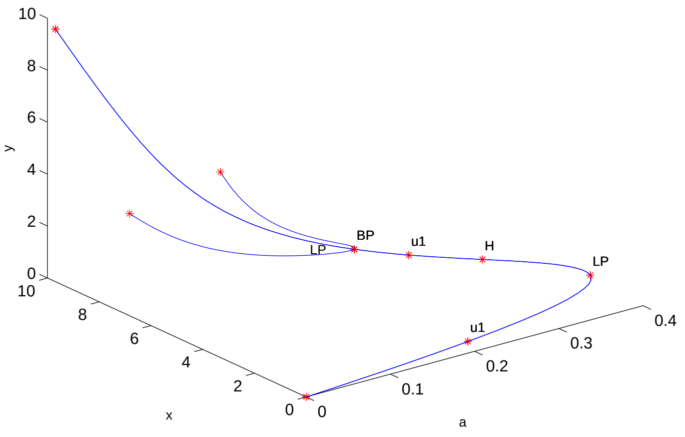


Figure 10: Equilibrium curve of `bratu.m`

8 The Brusselator example: Continuation of a solution to a boundary value problem in a free parameter

Discretized solutions of PDEs can also be continued in CL_MatCont. We illustrate this by continuing the equilibrium solution to a one-dimensional PDE. The curve type is called 'pde_1'.

The Brusselator is a system of equations intended to model the Belusov-Zhabotinsky reaction. This is a system of reaction-diffusion equations that is known to exhibit oscillatory behavior. The unknowns are the concentrations $X(x,t), Y(x,t), A(x,t)$ and $B(x,t)$ of four reactants. Here t denotes time and x is a one-dimensional space variable normalized so that $x \in [0,1]$. The length L of the reactor is a parameter of the problem. In our simplified setting A and B are constants.

The system is described by two partial differential equations:

$$\begin{array}{rcl} \frac{\partial X}{\partial t} & = & \frac{D_x}{L^2}\frac{\partial^2 X}{\partial x^2} + A - (B-1)X + X^2Y \\ \frac{\partial Y}{\partial t} & = & \frac{D_y}{L^2}\frac{\partial^2 Y}{\partial x^2} + BX - X^2Y \end{array} \tag{44}$$

with $x \in [0,1],\ t \geq 0$. Here D_x, D_y are the diffusion coefficients of X and Y. At the boundaries $x=0$ and $x=1$ Dirichlet conditions are imposed:

$$\begin{cases} X(0,t) = X(1,t) = A \\ Y(0,t) = Y(1,t) = \frac{B}{A} \end{cases} \tag{45}$$

We are interested in equilibrium solutions $X(x)$ and $Y(x)$ to the system and their dependence on the parameter L.

The approximate equilibrium solution is:

$$\begin{cases} X(x) & = & A + 2\sin(\pi x) \\ Y(x) & = & \frac{B}{A} - \frac{1}{2}\sin(\pi x) \end{cases} \tag{46}$$

The initial values of the parameters are: $A = 2$, $B = 4.6$, $D_x = 0.0016$, $D_y = 0.08$ and $L = 0.06$. The initial solution (46) is not an equilibrium, but the continuer will try to converge to an equilibrium close to the initial solution. We use equidistant meshes. To avoid spurious solutions (solutions that are induced by the discretization but do not actually correspond to solutions of the undiscretized problem) one can vary the number of mesh points by setting the parameter N. If the same solution is found for several discretizations, then we can assume that they correspond to solutions of the continuous problem.

The second order space derivative is approximated using the well-known three-points difference formula: $\frac{\partial^2 f}{\partial x^2} = \frac{1}{h^2}\left(f_{i-1} - 2f_i + f_{i+1}\right)$, where $h = \frac{1}{N+1}$, where N is the number of grid points on which we discretize X and Y. So N is a parameter of the problem and $2N$ is the number of state variables (which is not fixed in this case).

The Jacobian is a sparse 5-band matrix. In the ode-file describing the problem the Jacobian is introduced as a sparse matrix. The Hessian is never computed as such but second order derivatives are computed by finite differences whenever needed. We note that MATLAB 6.5 or 7 does not provide sparse structures for 3-dimensional arrays.

bruss.m

```
function out = bruss
%
% Odefile of 1-d Brusselator model
%

out{1} = @init;
out{2} = @fun_eval;
out{3} = @jacobian;
out{4} = @jacobianp;
out{5} = [];%@hessians;
out{6} = [];%@hessiansp;
out{7} = [];
out{8} = [];
out{9} = [];



% --------------------------------------------------------------------------


function dfdt = fun_eval(t,y,N,L)

x = y(1:N);
y = y(N+1:2*N);

A  = 2;
B  = 4.6;
Dx = 0.0016;
Dy = 0.008;
x0 = A; x1 = A;
y0 = B/A; y1 = B/A;
L2 = L^2;
h  = 1/(N+1);
cx = (Dx/L2)/(h*h);
cy = (Dy/L2)/(h*h);

dxdt = zeros(N,1);
dydt = zeros(N,1);

dxdt(1) = (x0-2*x(1)+x(2))*cx + A - (B+1)*x(1) + x(1)*x(1)*y(1);
dxdt(N) = (x(N-1)-2*x(N)+x1)*cx + A - (B+1)*x(N) + x(N)*x(N)*y(N);

dydt(1) = (y0-2*y(1)+y(2))*cy + B*x(1) - x(1)*x(1)*y(1);
dydt(N) = (y(N-1)-2*y(N)+y1)*cy + B*x(N) - x(N)*x(N)*y(N);

for i=2:N-1
  dxdt(i) = (x(i-1)-2*x(i)+x(i+1))*cx + A - (B+1)*x(i) + x(i)*x(i)*y(i);
  dydt(i) = (y(i-1)-2*y(i)+y(i+1))*cy + B*x(i) - x(i)*x(i)*y(i);
end

dfdt = [dxdt; dydt];

% --------------------------------------------------------------------------

function [tspan,y0,options] = init(N)
tspan = [0; 10];
A  = 2;
```

```
B  = 4.6;

y0 = zeros(2*N,1);

for i=1:N
  y0(i)   = A + 2*sin(pi*i/(N+1));
  y0(N+i) = B/A - 0.5*sin(pi*i/(N+1));
end
handles = feval(@bruss);
options = odeset('Vectorized','on', 'Jacobian', handles(3), 'JacobianP', handles(4));

% --------------------------------------------------------------------------

function dfdxy = jacobian(t,y,N,L)
x = y(1:N);
y = y(N+1:2*N);
A  = 2;
B  = 4.6;
Dx = 0.0016;
Dy = 0.008;
x0 = A; x1 = A;
y0 = B/A; y1 = B/A;
L2 = L^2;
h  = 1/(N+1);
cx = (Dx/L2)/(h*h);
cy = (Dy/L2)/(h*h);


%
% Sparse jacobian
%
A=zeros(2*N,3);
A(1:N-1,2)=cx;
A(1:N,3)=-2*cx -(B+1) + 2*x(1:N).*y(1:N);
A(1:N,4)=cx;

A(N+1:2*N,2) = cy;
A(N+1:2*N,3) = -2*cy -x(:).*x(:);
A(N+2:2*N,4) = cy;


A(1:N,1) = B - 2*x(:).*y(:);
A(N+1:2*N,5) = x(:).*x(:);

dfdxy = spdiags(A, [-N,-1:1,N] , 2*N, 2*N);
return

%
% Full matrix
%
dfxdx = zeros(N,N);
dfydy = zeros(N,N);
dfxdy = zeros(N,N);
dfydx = zeros(N,N);

for i=1:N
  if i>1, dfxdx(i,i-1) = cx; end;
```

```
          dfxdx(i,i)   = -2*cx -(B+1) + 2*x(i)*y(i);
  if i<N, dfxdx(i,i+1) = cx; end;
  dfxdy(i,i) = x(i)*x(i);

  if i>1, dfydy(i,i-1) = cy; end;
          dfydy(i,i)   = -2*cy -x(i)*x(i);
  if i<N, dfydy(i,i+1) = cy; end;
  dfydx(i,i) = B - 2*x(i)*y(i);
end

dfdxy = [ dfxdx, dfxdy; dfydx, dfydy ];

% ----------------------------------------------------------------------------

function dfdp = jacobianp(t,y,N,L)
x = y(1:N);
y = y(N+1:2*N);
A  = 2;
B  = 4.6;
Dx = 0.0016;
Dy = 0.008;
x0 = A; x1 = A;
y0 = B/A; y1 = B/A;
L2 = L^2;
h  = 1/(N+1);
cx = (Dx/L2)/(h*h);
cy = (Dy/L2)/(h*h);
kx = (-2/L)*cx;
ky = (-2/L)*cy;

Sx = zeros(N,1);
Sy = zeros(N,1);

Sx(1) = kx*(x0-2*x(1)+x(2));
Sy(1) = ky*(y0-2*y(1)+y(2));

Sx(N) = kx*(x(N-1)-2*x(N)+x1);
Sy(N) = ky*(y(N-1)-2*y(N)+y1);

i=2:N-1;
Sx(i)   = kx*(x(i-1)-2*x(i)+x(i+1));
Sy(i) = ky*(y(i-1)-2*y(i)+y(i+1));

dfdp = [ zeros(2*N,1) [Sx;Sy] ];

% ----------------------------------------------------------------------------
```

bruss.m

The model is implemented with 2 parameters: N and L; the values of A, B, D_x, D_y are hard-coded. Note that N is a parameter that cannot vary during the continuation. Therefore it does not have entries in Jacobianp. We should let the pde_1 curve know that *bruss.m* is the active file, the initial values of the parameters N and L are respectively 20 and 0.06 and the active parameter is L, i.e. the second parameter of `bruss.m`. So, first of all we have to get the approximate equilibrium solution which is provided in a subfunction 'init' of 'bruss.m'.

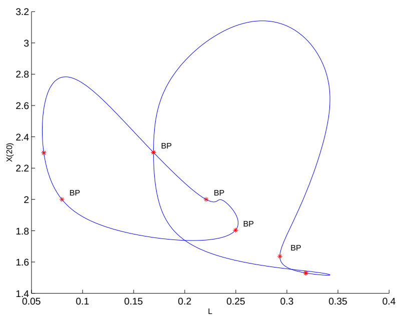


Figure 11: Equilibrium curves of `bruss.m`

We first locate the handle to the subfunction init and call it.

```
>N = 20; L = 0.06;
>handles = feval(@bruss);
>[t,x0,options] = feval(handles{1},N);
```

It sets the number of state variables to $2N$ and makes an initial vector $x0$ of length $2N$ containing the values of the approximate equilibrium solution. Now we inform the pde_1 curve that the second parameter of `bruss.m` is the active parameter and what the default values of the other parameters are. We also set some options.

```
>[x1,v1] = init_EP_EP(@bruss,x0,[N;L], [2]);
>opt = contset;opt=contset(opt,'MinStepsize', 1e-5);
>opt=contset(opt,'MaxCorrIters', 10);
>opt=contset(opt,'MaxNewtonIters', 20);
>opt=contset(opt,'FunTolerance', 1e-3);
>opt=contset(opt,'Singularities',1);
>opt=contset(opt,'MaxNumPoints',500);
>opt=contset(opt,'Locators',[]);
```

We start the continuation process by the statement
`[x,v,s,h] = cont(@pde_1,x1,v1,opt).`
In this case the number of state variables can be a parameter and the Jacobian can be sparse. The routine `cpl` can be used to plot two or three components of the curve.

9 Continuation of limit cycles

9.1 Mathematical definition

Consider the following differential equation

$$\frac{du}{dt} = f(u, \alpha) \tag{47}$$

with $u \in \mathbf{R}^n$ and $\alpha \in \mathbf{R}$. A periodic solution with period T satisfies the following system

$$\left\{ \begin{array}{l} \frac{du}{dt} = f(u, \alpha) \\ u(0) = u(T) \end{array} \right. . \tag{48}$$

For simplicity the period T is treated as a parameter resulting in the system

$$\left\{ \begin{array}{l} \frac{du}{d\tau} = T\, f(u, \alpha) \\ u(0) = u(1) \end{array} \right. . \tag{49}$$

If $u(\tau)$ is its solution then the shifted solution $u(\tau + s)$ is also a solution to (49) for any value of s. To select one solution, a phase condition is added to the system. The complete BVP (boundary value problem) is

$$\left\{ \begin{array}{rl} \frac{du}{d\tau} - Tf(u, \alpha) & = 0 \\ u(0) - u(1) & = 0 \\ \int_0^1 \langle u(t), \dot{u}_{old}(t) \rangle dt & = 0 \end{array} \right. \tag{50}$$

where $\dot{u}_{old}$ is the derivative of a previous solution. A limit cycle is a closed phase orbit corresponding to this periodic solution. This system is discretized using orthogonal collocation [6], the same way as it was done in AUTO [9]. The left hand side of the resulting system is the defining function $F(u, T, \alpha)$ for limit cycles.

9.2 Bifurcations

On a limit cycle curve the following bifurcations can occur

- *Branch Point of Cycles*, this will be denoted as `BPC`
- *Period Doubling*, denoted as `PD`
- *Fold*, also known as *Limit Point of Cycles*, this will be denoted as `LPC`
- *Neimark-Sacker*, this will be denoted as `NS`

The test function for the Period Doubling bifurcation is defined by the following system

$$\left\{ \begin{array}{rl} \dot{v}(\tau) - Tf_u(u, \alpha)v(\tau) + G\varphi(\tau) & = 0 \\ v(0) + v(1) & = 0 \\ \int_0^1 \langle \psi(\tau), v(\tau) \rangle d\tau & = 1 \end{array} \right. \tag{51}$$

here φ and ψ are so-called bordering vector-functions [23], see [12] for details on the implementation. The system is discretized using orthogonal collocation and solved using the standard MATLAB sparse system solver. The solution component $G \in \mathbf{R}$ of this system is the test function and equals zero when there is a Period Doubling bifurcation.

The Fold bifurcation is detected in the same way as the Fold bifurcation of equilibria, the last component of the tangent vector (the α component) is used as the test function.

The Neimark-Sacker bifurcation is detected by monitoring the eigenvalues of the monodromy matrix for the cycle. The monodromy matrix is computed like in AUTO by a block elimination in the discretized form of the Jacobian of (50).

BPC cycles are not generic in families of limit cycles, but they are common in the case of symmetries, if the branch parameter is also the continuation parameter. CL_MatCont uses a strategy that requires only the solution of linear systems; it is based on the fact that in a symmetry-breaking BPC cycle M_D has rank defect two, where M_D is the square matrix M_D, obtained from the discretized form of the Jacobian of (50). To be precise, if $h \in \mathcal{C}^1([0,1], \mathbb{R}^n)$, then

$$Mh = \begin{bmatrix} \dot{h} - Tf_x(x(t), \alpha)h \\ h(0) - h(1) \end{bmatrix},$$

and

$$M_D(h)_{dm} = \begin{bmatrix} (\dot{h} - Tf_x(x(t), \alpha)h)_{dc} \\ h(0) - h(1) \end{bmatrix},$$

where $()_{dm}$ and $()_{dc}$ denote discretization in mesh points and in collocation points, respectively. Therefore we border M_D with two additional rows and columns to obtain

$$M_{Dbb} = \begin{bmatrix} M_D & w_1 & w_2 \\ v_1^* & 0 & 0 \\ v_2^* & 0 & 0 \end{bmatrix},$$

so that M_{Dbb} is nonsingular in the BPC cycle. Then we solve the systems

$$M_{Dbb} \begin{bmatrix} \psi_{11} & \psi_{12} \\ g_{BPC11} & g_{BPC12} \\ g_{BPC21} & g_{BPC22} \end{bmatrix} = \begin{bmatrix} 0_{(Nm+1)n} & 0_{(Nm+1)n} \\ 1 & 0 \\ 0 & 1 \end{bmatrix},$$

where ψ_{11}, ψ_{12} have $(Nm+1)n$ components, and g_{BPC11}, g_{BPC12}, g_{BPC21}, and g_{BPC22}, are scalar test functions for the BPC. In the BPC cycle they all vanish.

The singularity matrix is

$$S = \begin{pmatrix} 0 & 0 & 0 & 0 & - & - & - & - \\ - & - & - & - & - & 0 & - & - \\ - & - & - & - & - & - & 0 & - \\ - & - & - & - & 1 & - & 1 & 0 \end{pmatrix} \qquad (52)$$

The first row corresponds to the BPC. It contains 4 zeros which indicates that g_{BPC11}, g_{BPC12}, g_{BPC21}, and g_{BPC22} should vanish. The last row corresponds to the NS. Because we have to exclude that all four testfunctions of the BPC are zeros, we introduce an extra testfunction which corresponds to the norm of these four testfunctions. A NS is detected if this norm is nonzero,the testfunction for the fold is nonzero and the testfunction for the NS is equal to zero.

9.2.1 Branch Point Locator

The location of BPC points in the non-generic situation (i.e. where some symmetry is present) as zeros of the test functions is numerically suspect because no local quadratic convergence can be guaranteed. This difficulty can be avoided by introducing an additional unknown $\beta \in \mathbf{R}$ and considering the minimally extended system:

$$\left\{\begin{array}{ll} \frac{dx}{dt} - Tf(x,\alpha) + \beta p_1 & = 0 \\ x(0) - x(1) + \beta p_2 & = 0 \\ \int_0^1 \langle x(t), \dot{x}_{old}(t)\rangle dt + \beta p_3 & = 0 \\ G[x,T,\alpha] & = 0 \end{array}\right. \tag{53}$$

where G is defined as in (87) and $[p_1^T p_2^T p_3]^T$ is the bordering vector $[w_{01}; w_{02}; w_{03}]^T$ in (89). We solve this system with respect to x, T, α and β by Newton's method with initial $\beta = 0$. A branch point (x, T, α) corresponds to a regular solution $(x, T, \alpha, 0)$ of system (53) (see [3],p. 165). We note, however that the second order partial derivatives (Hessian) of f with respect to x and α are required. The tangent vector v_{1st} at the BPC singularity is approximated as $v_{1st} = \frac{v_1+v_2}{2}$ where v_1 is the tangent vector in the continuation point previous to the BPC and v_2 is the one in the next point.

9.2.2 Processing

It is know that the periodic normal form at the Limit Point of Cycles (LPC) bifurcation is

$$\left\{\begin{array}{lcl} \frac{d\tau}{dt} & = & 1 - \xi + a\xi^2 + \cdots, \\ \frac{d\xi}{dt} & = & b\xi^2 + \cdots, \end{array}\right. \tag{54}$$

where $\tau \in [0, T]$, ξ is a real coordinate on the center manifold that is transverse to the limit-cycle, $a, b \in \mathbb{R}$, and dots denote nonautonomous T-periodic $O(\xi^3)$-terms. For each detected LPC point the normal form coefficient b is computed. A Cusp Point of Cycles (CPC) is detected for $b = 0$.

The periodic normal form at the Period Doubling (PD) bifurcation is

$$\left\{\begin{array}{lcl} \frac{d\tau}{dt} & = & 1 + a\xi^2 + \cdots, \\ \frac{d\xi}{dt} & = & c\xi^3 + \cdots, \end{array}\right. \tag{55}$$

where $\tau \in [0, 2T]$, ξ is a real coordinate on the center manifold that is transverse to the limit cycle, $a, c \in \mathbb{R}$, and dots denote nonautonomous $2T$-periodic $O(\xi^4)$-terms. The coefficient c determines the stability of the period doubled cycle in the center manifold and is computed during the processing of each PD point.

The periodic normal form at the Neimark-Sacker (NS) bifurcation is

$$\left\{\begin{array}{lcl} \frac{d\tau}{dt} & = & 1 + a|\xi|^2 + \cdots, \\ \frac{d\xi}{dt} & = & \frac{i\theta}{T}\xi + d\xi|\xi|^2 + \cdots, \end{array}\right. \tag{56}$$

where $\tau \in [0, T]$, ξ is a complex coordinate on the center manifold that is complementary to τ, $a \in \mathbb{R}, d \in \mathbb{C}$, and dots denote nonautonomous T-periodic $O(|\xi|^4)$-terms. The critical coefficient d in the periodic normal form for the NS bifurcation is computed during the processing of a NS point. The critical cycle is stable within the center manifold if Re $d < 0$ and is unstable if Re $d > 0$. In the former case, the Neimark-Sacker bifurcation is supercritical, while in the latter case it is subcritical.

9.3 Limitcycle initialization

For limit cycles the same problems occur as with equilibria, a limit cycle continuation can't be done by just calling the continuer as

```
[x,v,s,h,f]=cont(@limitcycle, x0, v0, opt)
```

The limit cycle curve file has to know

- which ode file to use,
- which parameter is active,
- the values of all parameters,
- the number of mesh and collocation points to use for the discretization.

Also an initial cycle `x0` has to be known. All this information can be supplied using any of the following three starting functions. All three return an initial cycle `x0` as well as its tangent vector `v0`.

- `[x0,v0]=init_H_LC(@odefile, x, p, ap, h, ntst, ncol)`
  Calculates an initial cycle from a Hopf point detected on an equilibrium curve. Here `odefile` is the ode-file to be used. `x` is a vector containing the values of the state variables returned by a previous equilibrium curve continuation. `p` is the vector containing the current values of the parameters. `ap` is the active parameter, `h` contains the value of the initial amplitude and `ntst` and `ncol` are the number of mesh and collocation points to be used for the discretization.

- `[x0,v0]=init_BPC_LC(@odefile, x, v, s, ap, ntst, ncol,h)`
  Calculates an initial cycle for starting a secondary cycle from a BPC detected on a previous calculated limit cycle. `odefile`, `ap`, `ntst` and `ncol` are the same as for `init_H_LC`. `x`, `v` and `s` are here the `x`, `v` and `s` as returned by a previous limit cycle continuation and `h` contains the value of the initial amplitude.

- `[x0,v0]=init_LC_LC(@odefile, x, v, s, par, ap, ntst, ncol)`
  This starter can be used to start a limit cycle continuation from a previous limit cycle continuation. `odefile`, `ap`, `ntst` and `ncol` are the same as for `init_H_LC`. `x` and `v` are here the `x` and `v` as returned by a previous limit cycle continuation. `s` is the special point structure for the point from where to start the new continuation. `par` is the parameter vector that is to be used for the new continuation. If this is the same as in the cycle returned by the continuer (arguably the natural situation), then this is the same as `s.data.parametervalues`

- `[x0,v0]=init_PD_LC(@odefile, x, s, ap, ntst, ncol)`
  This starter calculates an initial double period cycle from a period doubling bifurcation. All arguments are the same as for `init_LC_LC` except for `v` and `par`, which are not needed.

9.4 Example

For this example the following system from adaptive control was used in a feedback control system, as described in [17], [18] and further used in [23] (Example 5.4, p. 178):

$$\begin{cases} \dot{x} &= y \\ \dot{y} &= z \\ \dot{z} &= -\alpha z - \beta y - x + x^2 \end{cases} \tag{57}$$

This system has a Hopf point at the origin for $\alpha = 1$ and $\beta = 1$. From this Hopf point an initial cycle is calculated using the starter `init_H_LC`. The results of the continuation are plotted using the plot function `plotcycle(x,v,s,e)` (see Figure 12). This function plots the cycles `x`. `e` is an array whith either 2 or 3 elements for 2-dimensional and 3-dimensional plotting respectively. Its entries must be indices of state variables or active parameters in x. The index of the active parameter is `size(x,1)`

```
>> init;
>> [x0,v0]=init_EP_EP(@adapt,[0;0;0],[-10;1],[1]);
>> opt = contset; opt = contset(opt,'Singularities',1);
>> [x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = H , x = ( 0.000000 0.000000 0.000000 1.000002 )
First Lyapunov coefficient = -3.000001e-001

elapsed time  = 0.3 secs
npoints curve = 300
>> x1=x(1:3,s(2).index);p=[x(end,s(2).index);1];
>> [x0,v0]=init_H_LC(@adapt,x1,p,[1],1e-6,20,4);
>> opt = contset(opt,'MaxNumPoints',200);
>> opt = contset(opt,'Multipliers',1);
>> opt = contset(opt,'Adapt',1);
>> [xlc,vlc,slc,hlc,flc]=cont(@limitcycle,x0,v0,opt);
first point found
tangent vector to first point found
Limit point cycle (period = 6.283185e+000, parameter = 1.000000e+000)
Normal form coefficient = -1.306379e+000
Branch Point cycle(period = 6.283185e+000, parameter = 9.999996e-001)
Period Doubling (period = 6.364071e+000, parameter = 6.303020e-001)
Normal form coefficient = -4.267675e-002
Neimark-Sacker (period = 6.433818e+000, parameter = 1.895460e-008)
Neutral saddle
Period Doubling (period = 6.364071e+000, parameter = -6.303020e-001)
Normal form coefficient = 4.268472e-002

elapsed time  = 27.6 secs
npoints curve = 200
>> plotcycle(xlc,vlc,slc,[size(xlc,1) 1 2]);
```

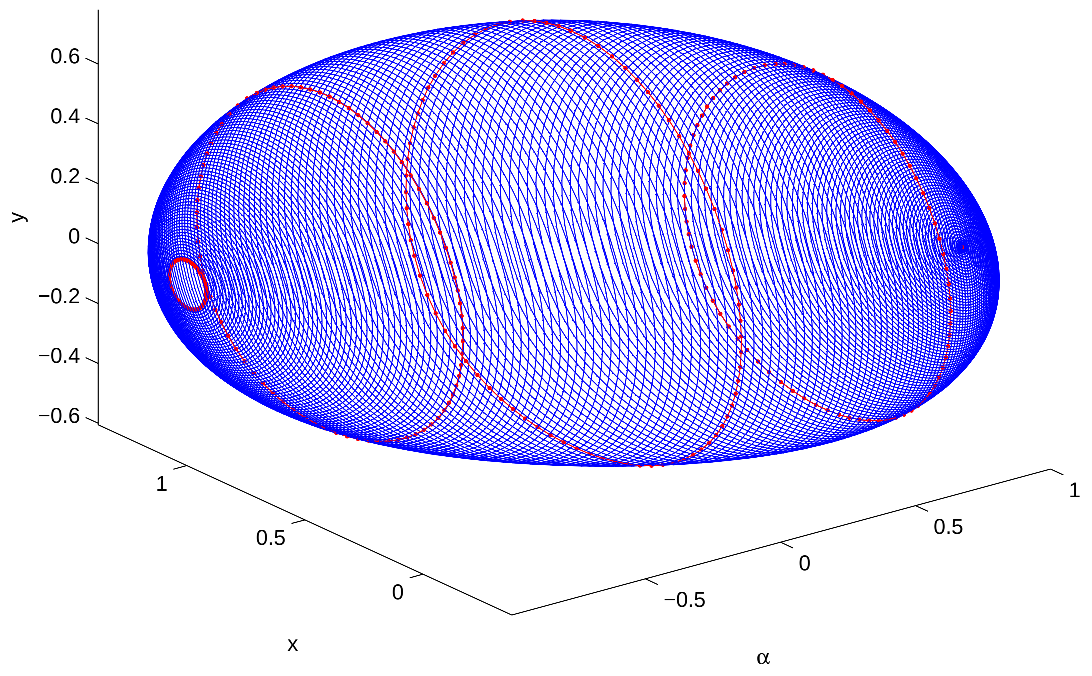


Figure 12: Computed limit cycle curve

The x-axis contains the active parameter , the y-axis the first state variable x and the z-axis the second state variable y. This run can be tested by the statement `testadapt1`. If you run only this example, do not forget to execute `init` statement first.

We note that the Limit point cycle and Branch point cycle detected in this run are degenerate: they reduce to the Hopf point itself. The Neimark-Sacker bifurcation is, in reality, a Neutral Saddle, i.e. it is an unstable periodic orbit with two real multipliers whose product is 1.

From the first Period Doubling bifurcation detected a limit cycle continuation of the nearby double period cycle is started. First, an initial cycle and its tangent vector are calculated using the starter `init_PD_LC`. The continuation is done using the standard continuer and the result is plotted using the `plotcycle` function (see figure 13).

```
>> [x1,v1]=init_PD_LC(@adapt,xlc,slc(4),40,4,1e-6);
>> opt=contset(opt,'MaxNumPoints',250);
>> [xlc2,vlc2,slc2,hlc2,flc2]=cont(@limitcycle,x1,v1,opt);
first point found
tangent vector to first point found
Branch Point cycle(period = 1.272814e+001, parameter = 6.303020e-001)
Period Doubling (period = 1.273437e+001, parameter = 5.796299e-001)
Normal form coefficient = -5.579636e-002
```

```
Neimark-Sacker (period = 1.154609e+001, parameter = 2.806142e-010)
Neutral saddle
Period Doubling (period = 1.106284e+001, parameter = -4.471966e-002)
Normal form coefficient = 6.970442e-003
Limit point cycle (period = 1.103168e+001, parameter = -4.494912e-002)
Normal form coefficient = 1.311327e+002
Neimark-Sacker (period = 1.076785e+001, parameter = -1.076152e-009)
Neutral saddle
Limit point cycle (period = 1.103169e+001, parameter = 4.494912e-002)
Normal form coefficient = -1.310582e+002
Period Doubling (period = 1.106284e+001, parameter = 4.471966e-002)
Normal form coefficient = -6.973392e-003
Neimark-Sacker (period = 1.154609e+001, parameter = 5.372279e-010)
Neutral saddle
Period Doubling (period = 1.273437e+001, parameter = -5.796298e-001)
Normal form coefficient = 5.580465e-002

elapsed time  = 62.3 secs
npoints curve = 250
>> plotcycle(xlc2,vlc2,slc2,[size(xlc2,1) 1 2]);
```

This run can be tested by the statement `testadapt2`.

9.5 The phase response curve

The phase response curve of a limit cycle, or PRC, is a curve, defined over the period of the cycle, that expresses, at each time of that period, the effect of a small input vector on the cycle. In experimental circumstances, this may correspond to injected current, to the addition of more chemical agents, etc. A positive value means that the current cycle is shortened in time, a negative value means that the period is prolonged.

The PRC, as it is generally computed, is exact for infinitesimally small input vectors. In practice the maximum norm of the input vector would depend on the needed accuracy and the values of the system's state variables.

The derivative phase response curve or dPRC also has some very important applications. For the concrete use of PRC and dPRC in synchronization studies in neural modeling, we refer to [20].

MatCont and Cl_MatCont support the computation of the PRC and dPRC of limit cycles during continuation, using the method described in [21]. The standard method, which uses numerical integration of the adjoint system, was implemented in XPPAUT [14].

The use in MatCont is easy: before starting the actual limit cycle continuation, the user can specify whether he wants to compute the PRC, dPRC or both, and he needs to indicate the input vector used. When a scalar is given as input, then the vector has this scalar as first entry and all other entries are zero. Then in separate plotting windows, for each computed step in limit cycle continuation, the PRC and/or dPRC are computed and plotted. The computed values are saved in the output f-array of the continuation.

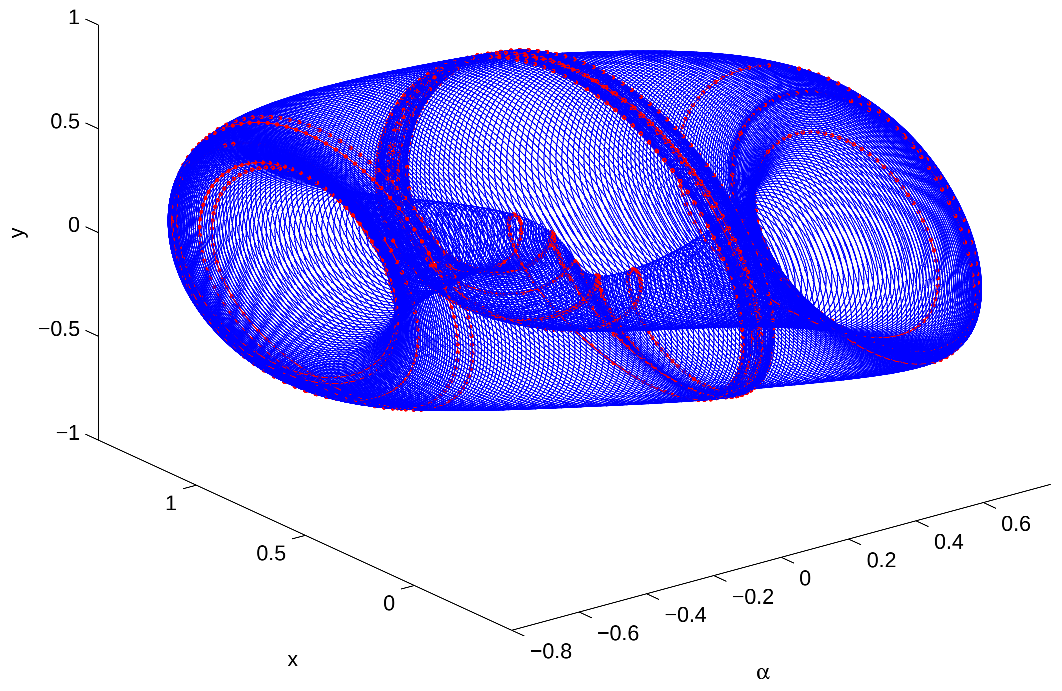


Figure 13: Computed limit cycle curve started from a Period Doubling bifurcation

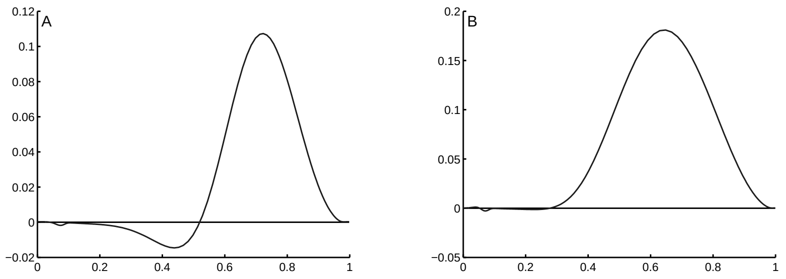


Figure 14: Two PRCs of limit cycles in the Morris-Lecar model [27], at different parameter-values.

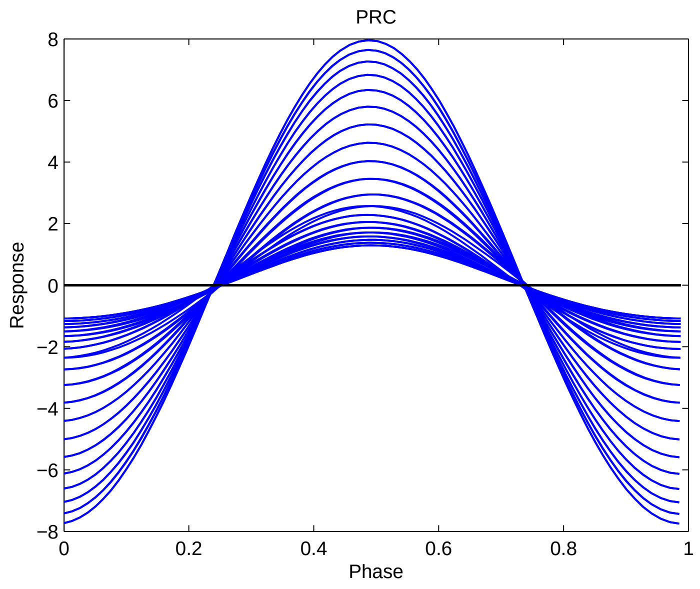
To illustrate the use in CL_MatCont, we here supply code that does a LC continuation experiment in the adapt-system (the same as above), but now also computes the PRC and dPRC of the system. The resulting figures are shown in Figure 15.

```
>> init;
>> [x0,v0]=init_EP_EP(@adapt,[0;0;0],[-10;1],[1]);
>> opt=contset;opt=contset(opt,'Singularities',1);
>> [x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = H , x = ( 0.000000 0.000000 0.000000 1.000000 )
First Lyapunov coefficient = -3.000000e-001

elapsed time  = 1.8 secs
npoints curve = 300
>> x1=x(1:3,s(2).index);p=[x(end,s(2).index);1];
>> [x0,v0]=init_H_LC(@adapt,x1,p,[1],1e-6,20,4);
>> opt = contset(opt,'MaxNumPoints',10);
>> opt = contset(opt,'Multipliers',1);
>> opt = contset(opt,'Adapt',1);
>> [xlc,vlc,slc,hlc,flc]=cont(@limitcycle,x0,v0,opt);
first point found
tangent vector to first point found
Limit point cycle (period = 6.283185e+000, parameter = 1.000000e+000)
Normal form coefficient = -1.240467e+000

elapsed time  = 3.0 secs
npoints curve = 10
>> par=slc(end).data.parametervalues;
>> [x1,v1] = init_LC_LC(@adapt, xlc, vlc, slc(end), par, 1, 20, 4);
>> opt = contset(opt,'PRC',1);
>> opt = contset(opt,'dPRC',1);
>> opt = contset(opt,'Input',1);
>> opt = contset(opt,'MaxNumPoints',20);
>> [xlc1,vlc1,slc1,hlc1,flc1]=cont(@limitcycle,x1,v1,opt);
```

As an example application for the specific computation during continuation, one can study the evolution of the PRCs of limit cycles, when these approach certain bifurcations. E.g. the closer a limit cycle is to a homoclinic orbit, the smaller the negative part of the PRC becomes (relatively): it is harder to increase the period of the cycle. In Figure 14 two PRCs are shown for the same dynamical system, but at different distances from a homoclinic orbit. In Figure 16 a sequence of PRCs is shown, computed during the continuation of a limit cycle approaching a homoclinic orbit. As in the MatCont case the computed values are saved in the output f-array of the continuation. See the testfiles `myml` and `mymlPRC`.



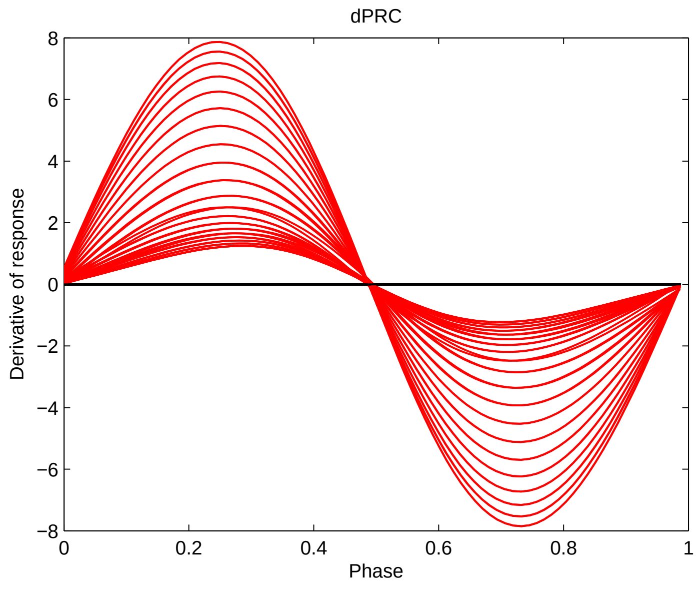


Figure 15: PRCs and dPRCs of the 'adapt'-system.

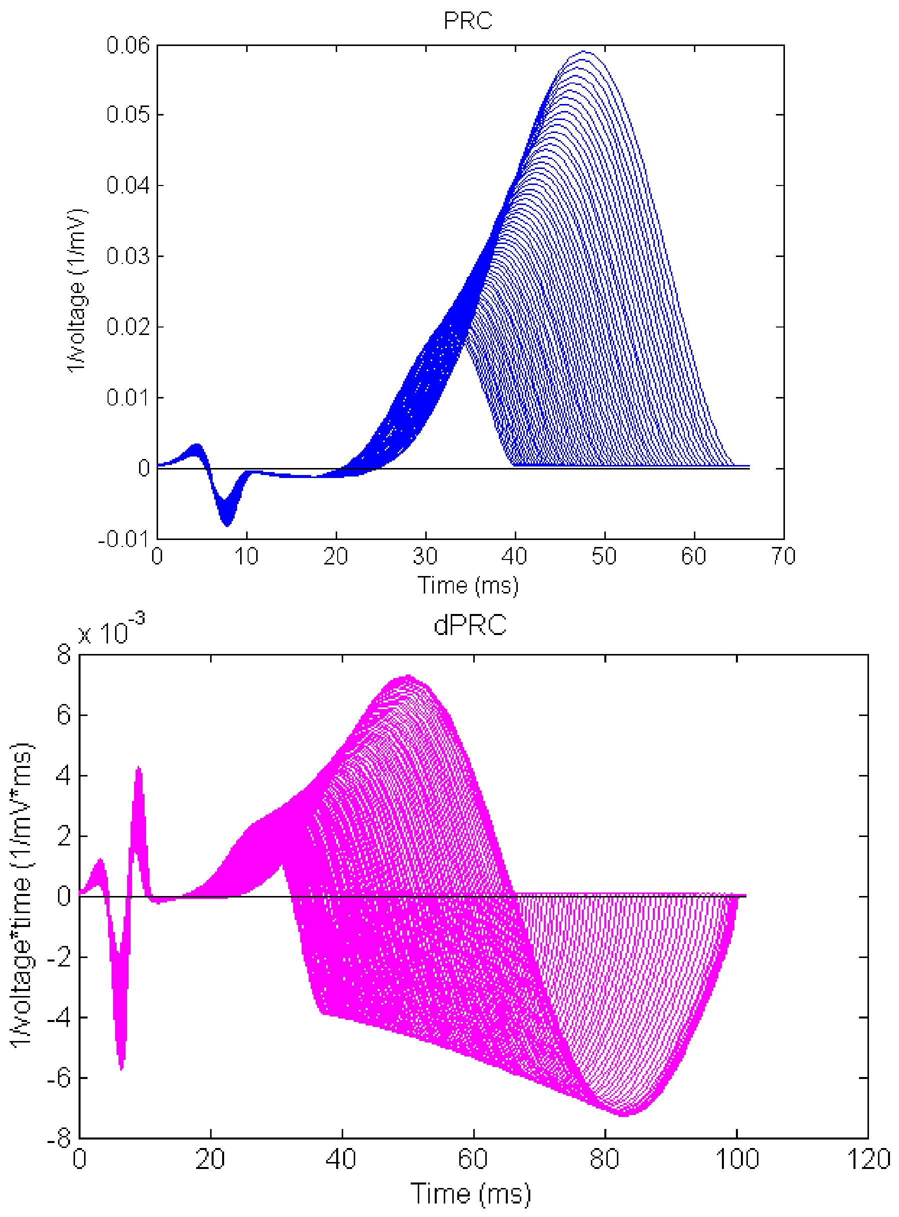


Figure 16: Top: PRCs of the Morris-Lecar system, computed during the limit cycle continuation. Bottom: the corresponding dPRCs.

10 Continuation of codim 1 bifurcations

10.1 Fold Continuation

10.1.1 Mathematical definition

In the toolbox fold curves are computed by *minimally extended defining systems* cf. [19], §4.1.2. The fold curve is defined by the following system

$$\left\{ \begin{array}{lcl} f(u,\alpha) & = & 0, \\ g(u,\alpha) & = & 0, \end{array} \right. \tag{58}$$

where $(u,\alpha) \in \mathbf{R}^{n+2}$, while g is obtained by solving

$$\begin{pmatrix} f_u(u,\alpha) & w_{bor} \\ v_{bor}^T & 0 \end{pmatrix} \begin{pmatrix} v \\ g \end{pmatrix} = \begin{pmatrix} 0_n \\ 1 \end{pmatrix}, \tag{59}$$

and $w_{bor}, v_{bor} \in \mathbf{R}^n$ are chosen such that the matrix in (59) is nonsingular. An advantage of this method is that the derivatives of g can be obtained easily from the derivatives of $f_u(u,\alpha)$:

$$g_z = -w^T (f_u)_z v$$

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

$$\begin{pmatrix} f_u^T(u,\alpha) & v_{bor} \\ w_{bor}^T & 0 \end{pmatrix} \begin{pmatrix} w \\ g \end{pmatrix} = \begin{pmatrix} 0_n \\ 1 \end{pmatrix}, \tag{60}$$

This method is implemented in the curve definition file `limitpoint.m`.

10.1.2 Bifurcations

In continuous-time systems there are four generic codim 2 bifurcations that can be detected along the fold curve:

- *Bogdanov - Takens.* We will denote this bifurcation by `BT`
- *Zero - Hopf* point, denoted by `ZH`
- *Cusp* point, denoted by `CP`
- *Branch* point, denoted by `BP`

To detect these singularities, we first define $bp+3$ test functions, where bp is the number of branch parameters:

- $\phi_1 = w^T v$ (cf. formula (39))
- $$\phi_2(u,\alpha) = \left(\left[\begin{array}{cc} (2f_u(u,\alpha) \odot I_n) & w_1 \\ v_1^T & d \end{array} \right] \backslash \begin{pmatrix} 0 \\ ... \\ 0 \\ 1 \end{pmatrix} \right)_{n+1} \tag{61}$$

- $\phi_3 = w^T f_{uu}[v, v]$
- $\phi_i = w^T f_{\beta_i}(u, \alpha)$

In these expressions v, w are the vectors computed in (59) and(60) respectively, $\odot$ is the bialternate matrix product, v_1, w_1 are $\frac{n(n-1)}{2}$ vectors chosen so that the square matrix in (61) is non-singular, and β_i (branch parameters) are components of α. The singularity matrix for $bp = 0$ is:

$$S = \begin{pmatrix} 0 & 0 & - \\ 1 & 0 & - \\ - & - & 0 \end{pmatrix} \tag{62}$$

The number of branch parameters is not fixed. If the number of branch parameters is 2 then this matrix has two more rows and columns. This singularity matrix is automatically extended:

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

10.1.3 Fold initialization

The only way to start a fold curve continuation in the toolbox is from a limit point. As in the case of equilibria, a fold curve continuation cannot be done by just calling the continuer as:
`[x,v,s,h,f]=cont(@limitpoint, x0, v0, opt)`.
The fold curve file has to know

- which odefile to use,
- which parameters are active,
- the values of all parameters.

To initialize the continuation one first gives the following statement:
`[x0,v0]=init_LP_LP(@odefile, xnew, p, ap` $(, bp)_{optional}$`)`.
In this statement `xnew` must be a vector that contains the values of the state variables. `p` must contain the current values of all the parameters and `ap` must be the indices of the 2 active parameters. In the most natural situation where `x` is the matrix returned by the previous equilibrium curve continuation one starts to build x_{new} by the statement `xnew=x(1:nphase,s(i).index)`, where `nphase` is the number of state variables and `s(i)` is the special point structure of the detected fold point on the equilibrium curve continuation. Next, the statement `p(ap_old)=x(end,s(i).index);` replaces the old value of the free parameter in the previous run by the newly found parameter `p`. `odefile` specifies the ode-file to be used. `bp` are the optional indices of the branch parameters. It also works without entering a value for this field.

10.1.4 Adaptation

It is possible to adapt the problem while generating the fold curve. This call updates the auxiliary variables used in the defining system of the computed branch. The bordering vectors v_{bor} and w_{bor} may require updating since they must at least be such that the matrices in (59) and (60) are nonsingular. Updating is done by replacing v_{bor} and w_{bor} by the normalized vectors v, w computed in (59) and (60) respectively.

10.1.5 Example

For this example the following system (a catalytic oscillator) is used

$$\left\{\begin{array}{lcl} \dot{x} & = & 2q_1z^2 - 2q_5x^2 - q_3xy \\ \dot{y} & = & q_2z - q_6y - q_3xy \\ \dot{z} & = & -q_4z - kq_4s \end{array}\right. \tag{63}$$

where $z = 1 - x - y - s$. The starting vector x0 and its tangent vector v0 are calculated from the following equilibrium curve continuation (q_2 is free).

```
>>  p=[2.5;1.4707;10;0.0675;1;0.1;0.4];ap1=[2];
[x0,v0]=init_EP_EP(@catalytic,[0.0029538; 0.76211;0.16781],p,ap1);
>> opt=contset;opt=contset(options,'MaxNumPoints',100);
>> opt=contset(opt,'MinStepSize',0.00001);
>> opt=contset(opt,'MaxStepSize',0.01);
>> opt=contset(opt,'Singularities',1);
>> [x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = H , x = ( 0.016358 0.523971 0.328337 1.051557 )
First Lyapunov coefficient = 1.070149e+001
label = LP, x = ( 0.024716 0.450260 0.375017 1.042049 )
a=1.166611e-001
label = LP, x = ( 0.054029 0.302241 0.459807 1.052200 )
a=-1.346501e-001
label = H , x = ( 0.077928 0.233065 0.492148 1.040992 )
First Lyapunov coefficient = 4.333093e+000

elapsed time  = 0.4 secs
npoints curve = 100
>> cpl(x,v,s,[4 1]);
```

The results are plotted using the plot function `cpl` where the fourth argument selects the fourth and first components of the solution which are the parameter q_2 and the coordinate x. The results can be seen in Figure 17. We start a forward and a backward fold continuation from the first LP detected on the previous equilibrium curve; q_2 and k are free in both runs.

```
>> x1=x(1:3,s(3).index);
>> p(ap1)=x(end,s(3).index);
>> [x0,v0]=init_LP_LP(@catalytic,x1,p,[2 7]);
```

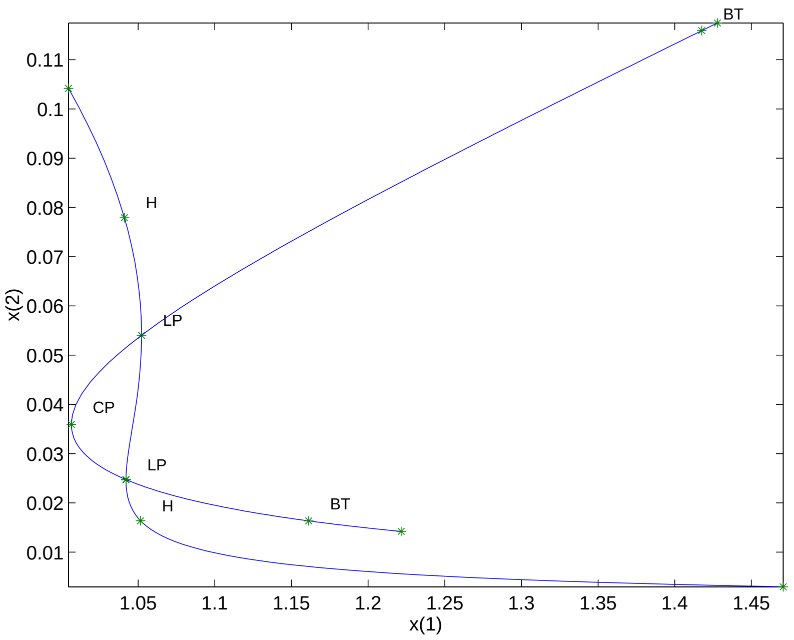


Figure 17: Computed fold curve

```
>> [x2,v2,s2,h2,f2]=cont(@limitpoint,x0,v0,opt);
first point found
tangent vector to first point found
label = CP, x = ( 0.035941 0.352005 0.451370 1.006408 0.355991 )
c=3.627923e-001
label = BT, x = ( 0.115909 0.315467 0.288437 1.417628 0.971398 )
(a,b)=(-8.378462e-002, -2.136282e+000)
elapsed time  = 0.5 secs
npoints curve = 100
>> hold on;cpl(x2,v2,s2,[4 1]);
>> opt=contset(opt,'Backward',1);
>> [x2,v2,s2,h2,f2]=cont(@limitpoint,x0,v0,opt);
first point found
tangent vector to first point found
label = BT, x = ( 0.016337 0.638410 0.200456 1.161199 0.722339 )
(a,b)=(-4.822563e-002, -1.937632e+000)
elapsed time  = 0.4 secs
npoints curve = 100
>> hold on;cpl(x2,v2,s2,[4 1]);
```

The results can be seen in Figure 17, where, as usual, the axes labels are added manually.

10.2 Hopf Continuation

10.2.1 Mathematical definition

In the MatCont / CL_MatCont toolbox Hopf curves are computed by *minimally extended defining systems*, cf. [19] §4.3.4. The Hopf curve is defined by the following system

$$\left\{ \begin{array}{rcl} f(u,\alpha) & = & 0, \\ g_{i_1 j_1}(u,\alpha,k) & = & 0 \\ g_{i_2 j_2}(u,\alpha,k) & = & 0 \end{array} \right. \tag{64}$$

with the unknowns $u, \alpha, k, (i_1, j_1, i_2, j_2) \in \{1,2\}$ and where $g = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}$ is obtained by solving

$$\begin{pmatrix} f_u^2 + kI_n & W_{bor} \\ V_{bor}^T & O \end{pmatrix} \begin{pmatrix} V \\ G \end{pmatrix} = \begin{pmatrix} 0_{n,2} \\ I_2 \end{pmatrix}, \tag{65}$$

where f_u has eigenvalues $\pm i\omega, \omega > 0$, $k = \omega^2$ and $V_{bor}, W_{bor} \in \mathbf{R}^{n\times 2}$ are chosen such that the matrix in (65) is nonsingular. $i_1, j_1, i_2, j_2, V_{bor}$ and W_{bor} are auxiliary variables that can be adapted. This method is implemented in the curve definition file `hopf.m`.

10.2.2 Bifurcations

In continuous-time systems there are four generic codim 2 bifurcations that can be detected along the Hopf curve:

- *Bogdanov - Takens.* We will denote this bifurcation by `BT`
- *Zero - Hopf* point, denoted by `ZH`
- *Double - Hopf* point, denoted by `HH`
- *Generalized Hopf* point, denoted by `GH`

To detect these singularities, we first define 4 test functions:

- $\phi_1 = k$
- $\phi_2 = \det(f_u)$
- $\phi_3 = \left(\begin{bmatrix} (2f_u(u,\alpha) \odot I_n) & w_1 \\ v_1^T & d \end{bmatrix} \setminus \begin{pmatrix} 0 \\ \dots \\ 0 \\ 1 \end{pmatrix} \right)_{n+1}$
- $\phi_4 = l_1$ (first Lyapunov coefficient, see (40)),

where v_1, w_1 are carefully constructed and updated $\frac{n(n-1)}{2} \times 2$ matrices.

In this case the singularity matrix is:

$$S = \begin{pmatrix} 0 & 0 & - & - \\ 1 & 0 & - & - \\ - & - & 0 & - \\ - & - & - & 0 \end{pmatrix} \tag{66}$$

10.2.3 Hopf initialization

The only way to start the continuation of a Hopf bifurcation curve is to start it from a Hopf point, typically found on an equilibrium curve. The continuation of Hopf points cannot simply be started by using the following statement:
`[x,v,s,h,f]=cont(@hopf, x0, v0, opt)`
The Hopf curve file has to know

- the values of all state variables,
- which ode file to use,
- which parameters are active,
- the values of all parameters.

Also an initial point `x0` has to be known. All this information can be supplied using the following statement:
`[x0,v0]=init_H_H(@odefile, xnew, p, ap).`

The input arguments `odefile, xnew, p, ap` are built exactly as in `init_LP_LP`. The output vector `x0` consists of x_{new} extended with the free parameters and paramter k from (65).

10.2.4 Adaptation

It is possible to adapt the problem while generating the Hopf curve. This call updates the auxiliary variables used in the defining system of the computed branch. The bordering matrices V_{bor} and W_{bor} may require updating since they must at least be such that the matrix in (65) is nonsingular. Updating of V_{bor} and W_{bor} is done exactly as in the LP case. Updating of i_1, j_1, i_2, j_2 is done in such a way that the linearized system of (64) is as well-conditioned as possible.

10.2.5 Example

For this example we again use the system (63). The starting vector is calculated from the first Hopf bifurcation point detected in the equilibrium continuation example (see 10.1.5). q_2 and k are free.

```
>> x1=x(1:3,s(2).index);
>> p(ap1)=x(end,s(2).index);
>> [x0,v0]=init_H_H(@catalytic,x1,p,[2 7]);
>> opt=contset;opt=contset(options,'Singularities',1);
>> opt=contset(opt,'MaxStepsize',0.1);
>> opt=contset(opt,'MaxNumPoints',300);
>> opt=contset(opt,'Backward',0);
>> [x2,v2,s2,h2,f2]=cont(@hopf,x0,[],opt);
first point found
tangent vector to first point found
label = GH, x = ( 0.018022 0.368277 0.497926 0.891362 0.232514 0.003324 )
l2=-7.768960e+002
```

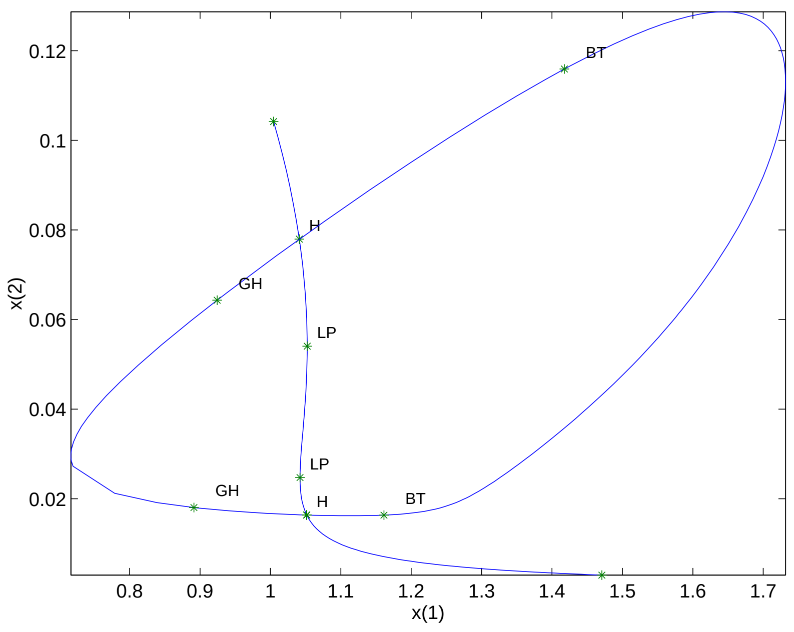


Figure 18: Computed Hopf curve

```
label = GH, x = ( 0.064312 0.211095 0.554869 0.924257 0.305881 0.003512 )
l2=-2.401240e+002
label = BT, x = ( 0.115909 0.315467 0.288437 1.417628 0.971398 -0.000000 )
(a,b)=(8.378437e-002, 2.136280e+000)
label = BT, x = ( 0.016337 0.638410 0.200456 1.161199 0.722339 0.000000 )
(a,b)=(-4.822564e-002, -1.937633e+000)
Closed curve detected at step 159

elapsed time  = 1.3 secs
npoints curve = 159
>> hold on;cpl(x2,v2,s2,[4 1]);
```

The results of the last computed curve are plotted using the standard plot function `cpl` where the fourth argument is used to select the fourth and first components of the solution which are the parameter q_2 and the coordinate x. The result can be seen in Figure 18.

10.3 Period Doubling

10.3.1 Mathematical definition

The Period Doubling bifurcation curve is defined by the following system

$$\begin{cases} \frac{du}{d\tau} - Tf(u,\alpha) &= 0 \\ u(0) - u(1) &= 0 \\ \int_0^1 \langle u(t), \dot{u}_{old}(t)\rangle dt &= 0 \\ G(u,T,\alpha) &= 0 \end{cases} \tag{67}$$

this is exactly the system defining limit cycles but with one extra constraint $G(u,T,\alpha) = 0$ where $G(u,T,\alpha)$ is defined as the solution component G of the system

$$\begin{cases} \dot{v}(\tau) - Tf_u(u,\alpha)v(\tau) + G\varphi(\tau) &= 0 \\ v(0) + v(1) &= 0 \\ \int_0^1 \langle \psi(\tau), v(\tau)\rangle d\tau &= 1 \end{cases} \tag{68}$$

which is exactly the same system as was used to detect the Period Doubling bifurcation.

Instead of using this functional G both systems can be combined in one larger system

$$\begin{cases} \frac{du}{d\tau} - Tf(u,\alpha) &= 0 \\ u(0) - u(1) &= 0 \\ \int_0^1 \langle u(t), \dot{u}_{old}(t)\rangle dt &= 0 \\ \dot{v}(\tau) - Tf_u(u,\alpha)v(\tau) &= 0 \\ v(0) + v(1) &= 0 \\ \int_0^1 \langle v_{old}(\tau), v(\tau)\rangle d\tau - 1 &= 0 \end{cases} \tag{69}$$

The first method (using system (67) and (68)) is implemented in the curve definition file `perioddoubling`.

10.3.2 Bifurcations

In MatCont / CL_MatCont there are four generic codim 2 bifurcations that can be detected along the flip curve:

- *Strong 1:2 resonance.* We will denote this bifurcation with `R2`
- *Fold - flip*, *Limit Point - Period Doubling*or We will denote this bifurcation with `LPPD`
- *Flip - Neimark-Sacker*, denoted as `PDNS`
- *Generalized period doubling* point, denoted as `GPD`

To detect these singularities, we define 4 test functions:

- $\phi_1 = (v_1^*)^T_{W_1} L_{C\times M} v_{1M}$
- $\phi_2 = (\psi_1^*)^T_{W_1} (F(u_{0,1}(t)))_C$
- $\phi_3 = det\,((M \odot M) - I_n)$
- $\phi_4 = c$

where c is the coefficient defined in (55), M is the monodromy matrix and $\odot$ is the bialternate product.

The v_1's and ψ_1's are obtained as follows. For a given $\zeta \in \mathcal{C}^1([0,1],\mathbb{R}^n)$ we consider three different discretizations:

- $\zeta_M \in \mathbb{R}^{(Nm+1)n}$ the vector of values in the mesh points,
- $\zeta_C \in \mathbb{R}^{Nmn}$ the vector of values in the collocation points,
- $\zeta_W = \begin{bmatrix} \zeta_{W_1} \\ \zeta_{W_2} \end{bmatrix} \in \mathbb{R}^{Nmn} \times \mathbb{R}^n$ where ζ_{W_1} is the vector of values in the collocation points multiplied with the Gauss - Legendre weights and the lengths of the mesh intervals, and $\zeta_{W_2} = \zeta(0)$.

Formally we further introduce $L_{C\times M}$ which is a structured sparse matrix that converts a vector ζ_M of values in the mesh points into a vector ζ_C of values in the collocation points by $\zeta_C = L_{C\times M}\zeta_M$. We compute v_{1M} by solving

$$\begin{bmatrix} D - TA(t) \\ \delta_0 + \delta_1 \end{bmatrix}_{disc} v_{1M} = 0. \tag{70}$$

The normalization of v_{1M} is done by requiring $\sum_{i=1}^{N}\sum_{j=0}^{m} \sigma_j \langle (v_{1M})_{(i-1)m+j}, (v_{1M})_{(i-1)m+j}\rangle \Delta_i = 1$ where σ_j is the Gauss-Lagrange quadrature coefficient and Δ_i is the length of the i-th interval. By discretization we obtain

$$(v_{1W}^*)^T \begin{bmatrix} D - TA(t) \\ \delta_0 + \delta_1 \end{bmatrix}_{disc} = 0.$$

To normalize $(v_1^*)_{W_1}$ we require $\sum_{i=1}^{N}\sum_{j=1}^{m} |((v_1^*)_{W_1})_{ij}|_1 = 1$. Then $\int_0^1 \langle v_1^*(t), v_1(t)\rangle dt$ is approximated by $(v_1^*)_{W_1}^T L_{C\times M} v_{1M}$ and if this quantity is nonzero, v_{1W}^* is rescaled so that $\int_0^1 \langle v_1^*(t), v_1(t)\rangle dt = \frac{1}{2}$. We compute ψ_{1W}^* by solving

$$(\psi_{1W}^*)^T \begin{bmatrix} D - TA(t) \\ \delta_0 - \delta_1 \end{bmatrix}_{disc} = 0$$

and normalize $\psi_{1W_1}^*$ by requiring $\sum_{i=1}^{N}\sum_{j=1}^{m} |((\psi_1^*)_{W_1})_{ij}|_1 = 1$. Then $\int_0^1 \langle \psi_1^*(t), F(u_{0,1}(t))\rangle dt$ is approximated by $(\psi_1^*)_{W_1}^T (F(u_{0,1}(t)))_C$ and if this quantity is nonzero, ψ_{1W}^* is rescaled so that $\int_0^1 \langle \psi_1^*(t), F(u_{0,1}(t))\rangle dt = 1$. a_1 can be computed as $(\psi_{W_1}^*)^T (B(t, v_{1M}, v_{1M}))_C$. The computation of $(h_{2,1})_M$ is done by solving

$$\begin{cases} (D - TA(t))_{C\times M}(h_{2,1})_M &= (B(t; v_{1M}, v_{1M}))_C + 2a_1(F(u_{0,1}(t)))_C, \ t \in [0,1] \\ (\delta(0) - \delta(1))(h_{2,1})_M &= 0, \\ (\psi_{W_1}^*)^T L_{C\times M}(h_{2,1})_M &= 0. \end{cases}$$

The expression for the normal form coefficient c becomes

$$\begin{aligned} c = \tfrac{1}{3}((v_{1W_1}^*)^T (C(t; v_{1M}, \tfrac{1}{T}v_{1M}, v_{1M})_C + 3(B(t; v_{1M}, (h_{2,1})_M)_C \\ -6\tfrac{a_1}{T}(v_{1W_1}^*)^T (A(t)v_1(t))_C). \end{aligned}$$

The singularity matrix is:

$$S = \begin{pmatrix} 0 & - & - \\ - & 0 & - \\ 1 & 1 & 0 \end{pmatrix}. \tag{71}$$

10.3.3 Period doubling initialization

The only way to start a Period Doubling bifurcation curve continuation supported in the current version is to start it from a Period Doubling bifurcation point on a limit cycle curve. This can be done using the following statement: `[x0,v0]=init_PD_PD(@odefile, x, s, ap, ntst, ncol)`. `x` should be the `x` as returned by the previous limit cycle continuation. `s` is the special point structure of the detected Period Doubling point on the limit cycle curve. `odefile` specifies the ode-file to be used. `ap` is the active parameter and `ntst` and `ncol` are again the number of mesh and collocation points for the discretization.

10.3.4 Example

For this example the following system is used

$$\begin{cases} \dot{x} &= y \\ \dot{y} &= z \\ \dot{z} &= -\alpha z - \beta y - x + x^2 \end{cases} \tag{72}$$

that is the same system as in the limit cycle continuation example. The starting vector `x0` is calculated from the Period Doubling bifurcation detected in the limit cycle example (section 9.4) using `init_PD_PD`. Continuation is done using a call to the standard continuer with `perioddoubling` as curve definition file. The results are plotted using the standard plot function `cpl` where the fourth argument is used to select the 245th and 246th component of the solution which are the parameters α and β. The results can be seen in Figure 19. The labels of the plot are added manually and the direction of the x-axis is reversed. It can be tested by the statement `testadapt3`.

```
>> [x0,v0]=init_PD_PD(@adapt,xlc,slc(3),[1 2],20,4);
>> opt = contset; opt = contset(opt,'Singularities',1);
>> [xpd,vpd,spd,hpd,fpd]=cont(@perioddoubling,x0,v0,opt);
first point found
tangent vector to first point found
1:2 resonance (period = 4.841835e+000, parameters = -3.400993e-010, 1.698711e+000)
1:2 resonance (period = 9.058318e+000, parameters = 4.486764e-009, 6.782783e-001)

elapsed time  = 154.4 secs
npoints curve = 300
>> cpl(xpd,vpd,spd,[245 246]);
```

10.4 Continuation of fold bifurcation of limit cycles

10.4.1 Mathematical definition

A Fold bifurcation of limit cycles (Limit Point of Cycles, LPC) generically corresponds to a turning point of a curve of limit cycles. It can be characterized by adding an extra constraint $G = 0$ to (50) where G is the Fold test function. The complete BVP defining a LPC point

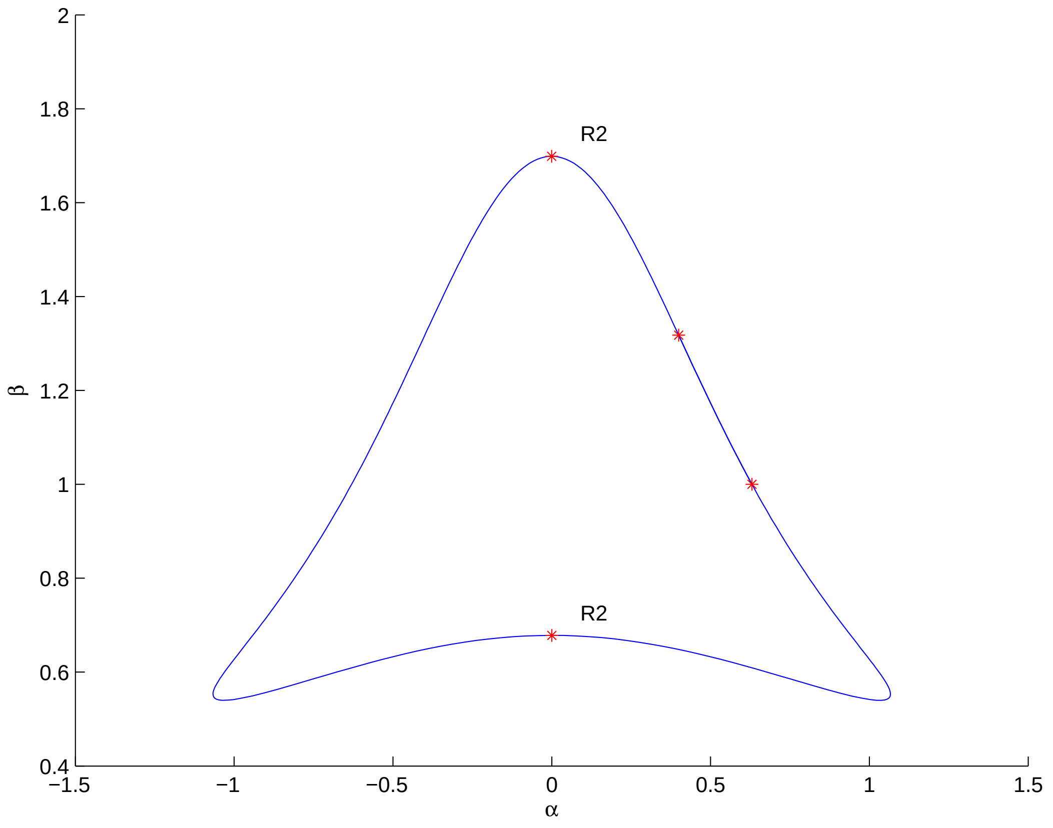


Figure 19: Computed Period Doubling curve

using the minimal extended system is

$$\begin{cases} \frac{du}{dt} - Tf(u,\alpha) & = 0 \\ u(0) - u(1) & = 0 \\ \int_0^1 \langle u(t), \dot{u}_{old}(t) \rangle dt & = 0 \\ G[u,T,\alpha] & = 0 \end{cases} \tag{73}$$

where G is defined by requiring

$$N^1 \begin{pmatrix} v \\ S \\ G \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \tag{74}$$

Here v is a function, S and G are scalars and

$$N^1 = \begin{bmatrix} D - Tf_u(u(t),\alpha) & -f(u(t),\alpha) & w_{01} \\ \delta_1 - \delta_0 & 0 & w_{02} \\ Int_{f(u(\cdot),\alpha)} & 0 & w_{03} \\ Int_{v_{01}} & v_{02} & 0 \end{bmatrix} \tag{75}$$

where the bordering functions v_{01}, w_{01}, vector w_{02} and scalars v_{02} and w_{03} are chosen so that N^1 is nonsingular [12]. This method (using system (74) and (75)) is implemented in the curve definition file `limitpointcycle`. The discretization is done using orthogonal collocation in the same way as it was done for limit cycles.

10.4.2 Bifurcations

In CL_MatCont there are five generic codim 2 bifurcations that can be detected along the fold curve:

- *Branch point.* We will denote this bifurcation with `BPC`
- *Strong 1:1 resonance.* We will denote this bifurcation with `R1`
- *Cusp* point, denoted by `CPC`
- *Fold - flip*, *Limit Point - Period Doubling*or We will denote this bifurcation by `LPPD`
- *Fold - Neimark-Sacker*, denoted by `LPNS`

A *Generalized Hopf* (`GH`) marks the end (or start) of an LPC curve.
To detect the generic singularities, we first define $bp+5$ test functions, where bp is the number of branch parameters:

- $\psi_i = w^T f_{\beta_i}(u,\alpha), i \in (1,...,bp)$
- $\psi_{bp+1} = (\varphi_1^*)_{W_1}^T L_{C\times M} v_{1M}$
- $\psi_{bp+2} = b$ (normal form coefficient)
- $\phi_{bp+3} = det\,(M + I_n)$

- $\psi_{bp+4} = det\left((M2 \odot M2) - I_n\right)$

In the ψ_i expressions w is the vector computed in (60) and β_i (branch parameter) is a component of α.

In the second expression ψ_{bp+1}, we compute v_{1M} by solving

$$\left[\begin{array}{c} D - TA(t) \\ \delta_0 - \delta_1 \\ \int_0^1 F(u_{0,1}(t))L_{C\times M} \end{array}\right]_{disc} v_{1M} = \left[\begin{array}{c} TF(u_{0,1}(t)) \\ 0 \\ 0 \end{array}\right]_{disc}. \tag{76}$$

By discretization we obtain

$$(\varphi^*_{1W})^T \left[\begin{array}{c} D - TA(t) \\ \delta_0 - \delta_1 \end{array}\right]_{disc} = 0.$$

To normalize $(\varphi_1^*)_{W_1}$ we require $\sum_{i=1}^{N}\sum_{j=1}^{m}\left|((v_1^*)_{W_1})_{(i-1)m+j}\right|_1 = 1$. Then $\int_0^1 \langle\varphi_1^*(t), v_1(t)\rangle dt$ is approximated by $(\varphi_1^*)^T_{W_1} L_{C\times M} v_{1M}$ and if this quantity is nonzero, φ^*_{1W} is rescaled so that $\int_0^1 \langle\varphi_1^*(t), v_1(t)\rangle dt = 1$.

So the third expression for the normal form coefficient b becomes

$$b = \frac{1}{2}((\varphi^*_{1W_1})^T((B(t; v_{1M}, v_{1M})_C + 2(A(t)v_1(t))_C)).$$

In the fourth expression, M is the monodromy matrix.

In the fifth expression, $M2 = (M - I_n)^2$, restricted to the subspace without the two eigenvalues with smallest norm.

The number of branch parameters is not fixed. If the number of branch parameters is 3 then this matrix has three more rows and columns. This singularity matrix is automatically extended:

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

10.4.3 Fold initialization

One way to start a Fold bifurcation of cycles continuation supported in the current version is to start it from a fold bifurcation point (LPC) on a limit cycle curve. This can be done using the following statement: `[x0,v0]=init_LPC_LPC(@odefile, x, s, ap, ntst, ncol`$(, bp)_{optional}$`)`. `x` should be the `x` as returned by the previous limit cycle continuation. `s` is the special point structure of the detected Fold bifurcation point on the limit cycle curve. `odefile` specifies the ode-file to be used. `ap` is the active parameter and `ntst` and `ncol` are again the number of mesh and collocation points for the discretization. `bp` are the optional indices of the branch parameters. It also works without entering a value for this field: `[x0,v0]=init_LPC_LPC(@odefile, x, s, ap, ntst, ncol)`.

10.4.4 Example

We consider the following system

$$\begin{cases} \dot{v} &= y - 0.5(v+0.5) - 2w(v+0.7) - m_\infty(v-1) \\ \dot{w} &= 1.15(w_\infty - w)\tau \end{cases} \tag{77}$$

where $m_\infty(v) = (1+\tanh((v+0.01)/0.15))/2$, $w_\infty(v) = (1+\tanh((v-z)/0.145))/2$ and $\tau = \cosh((v-0.1)/0.29)$. Here v and w are the state variables and y and z are the parameters. This is a modification of the fast subsystem of the Morris-Lecar equations studied in [30],[31]; the Morris-Lecar equations were introduced in [27] as a model for the electrical activity in the barnacle giant muscle fiber. In our model y corresponds to the slow variable in the fast Morris-Lecar equations; z is the potential for which $w_\infty(z) = \frac{1}{2}$.

We find a stable equilibrium (EP) for $y = 0.110472$ and $z = 0.1$ at $(0.04722, 0.32564)$ by time integration. We continue this equilibrium with free parameter y for decreasing values of y. The starting vector `x0` and its tangent vector `v0` are calculated from the following equilibrium curve continuation.

```
>> p=[0.11047;0.1];ap1=[1];
>> [x0,v0]=init_EP_EP(@MLfast,[0.047222;0.32564],p,ap1);
>> opt=contset;opt=contset(opt,'Singularities',1);
>> opt=contset(opt,'MaxNumPoints',65);
>> opt=contset(opt,'MinStepSize',0.00001);
>> opt=contset(opt,'MaxStepSize',0.01);
>> opt=contset(opt,'Backward',1);
>> [x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = H , x = ( 0.036756 0.294770 0.075659 )
First Lyapunov coefficient = 8.234573+000
label = LP, x = ( -0.033738 0.136501 -0.020727 )
a=1.036706e+001
label = H , x = ( -0.119894 0.045956 0.033207 )
Neutral saddle
label = LP, x = ( -0.244915 0.008514 0.083257 )
a=2.697414e+000

elapsed time  = 0.3 secs
npoints curve = 65
>> cpl(x,v,s,[3 1]);
```

We find a Hopf (H) bifurcation point at $y = 0.075659$, two limit points (LP) at $y = -0.020727$ and $y = 0.083257$ and a neutral saddle (H) at $y = 0.033207$. There are stable equilibria before the first H point and after the second LP point and unstable equilibria between the first H point and the second LP point. The Lyapunov coefficient in the first Hopf point $l_1 = 8.234573$ is positive which means that the periodic orbits are born unstable. The results are plotted using the plot function `cpl` where the fourth argument is used to select the third and first component of the solution which are the parameter y and the coordinate v. The results can be seen in Figure 20.

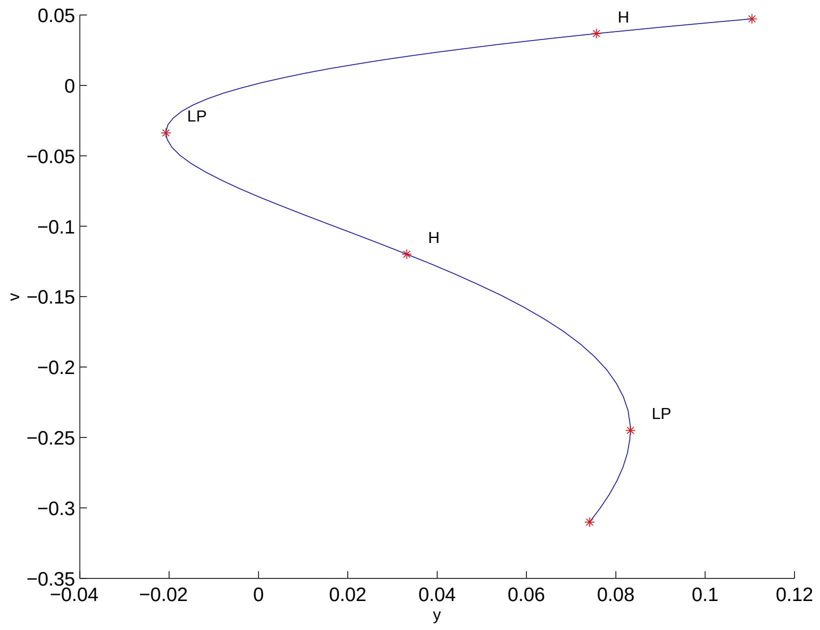


Figure 20: Computed equilibrium curve

The detected Hopf point is used to start a limit cycle continuation. We choose $N = 30$ test intervals and $m = 4$ collocation points for the discretization. The periodic orbit is initially unstable. We detect a limit point of cycles LPC at $y = 0.084569$. At this point the stability is gained. Afterwards the stability is preserved but the period tends to infinity and the periodic orbits end in a homoclinic orbit. The results can be seen in Figure 21.

```
>> x1=x(1:2,s(2).index);p=[x(end,s(2).index);0.1];
>> [x0,v0]=init_H_LC(@MLfast,x1,p,ap1,0.0001,30,4);
>> opt=contset;opt=contset(opt,'MaxNumPoints',65);
>> opt=contset(opt,'IgnoreSingularity',1);
>> opt=contset(opt,'Singularities',1);
>> [x2,v2,s2,h2,f2]=cont(@limitcycle,x0,v0,opt);
first point found
tangent vector to first point found
Limit point cycle (period = 4.222011e+000, parameter = 8.456948e-002)
Normal form coefficient = -2.334576e-001
elapsed time  = 38.1 secs
npoints curve = 500
>> plotcycle(x2,v2,s2,[1 2]);
```

This can be executed by running `fold1`.

The starting vector `x0` is calculated from the LPC on this branch using `init_LPC_LPC`. Continuation is done using a call to the standard continuer with `limitpointcycle` as curve definition file. We free both y and z to continue the LPC curve through this LPC point. The results are plotted using the standard plot function `plotcycle` where the fourth argument is used to select the coordinates. The results can be seen in Figure 22. We note that it shrinks

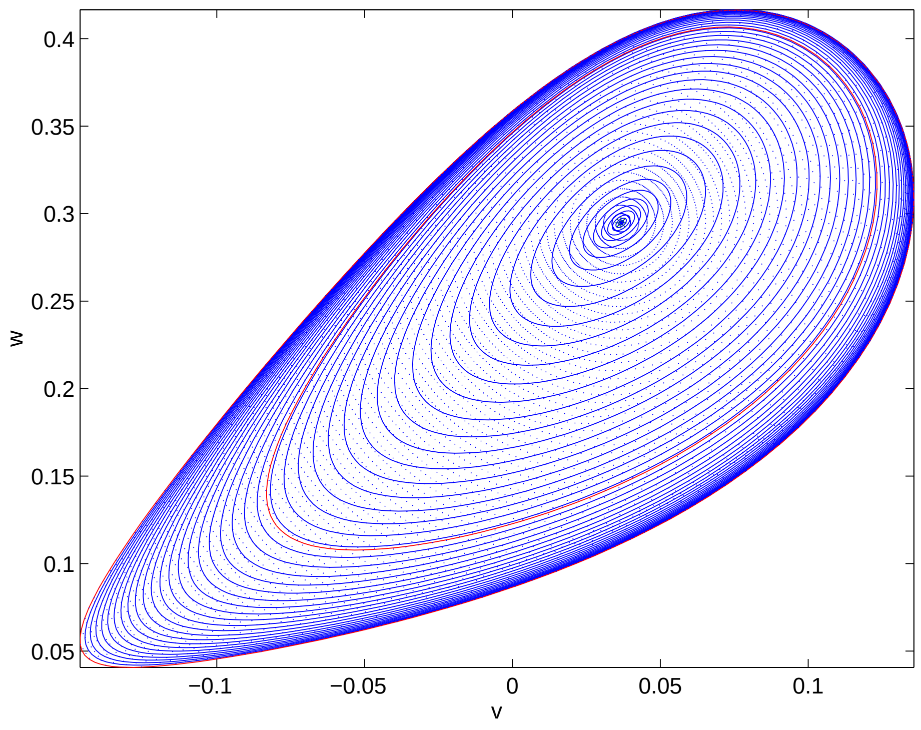


Figure 21: Computed limit cycle curve started from a Hopf bifurcation

to a single point. The labels of the plot are added manually .

```
>> [x0,v0]=init_LPC_LPC(@MLfast,x2,s2(3),[1 2],30,4);
>> opt=contset;opt=contset(opt,'MaxNumPoints',35);
>> opt=contset(opt,'Singularities',1);
>> [x3,v3,s3,h3,f3]=cont(@limitpointcycle,x0,v0,opt);
first point found
tangent vector to first point found
elapsed time  = 33.7 secs
npoints curve = 35
>> plotcycle(x3,v3,s3,[1 2]);
```

10.5 Continuation of torus bifurcation of limit cycles

10.5.1 Mathematical definition

A torus bifurcation of limit cycles (Neimark-Sacker, NS) generically corresponds to a bifurcation to an invariant torus, on which the flow contains periodic or quasi-periodic motion. It can be characterized by adding an extra constraint $G = 0$ to (50) where G is the torus test function which has four components from which two are selected. The complete BVP

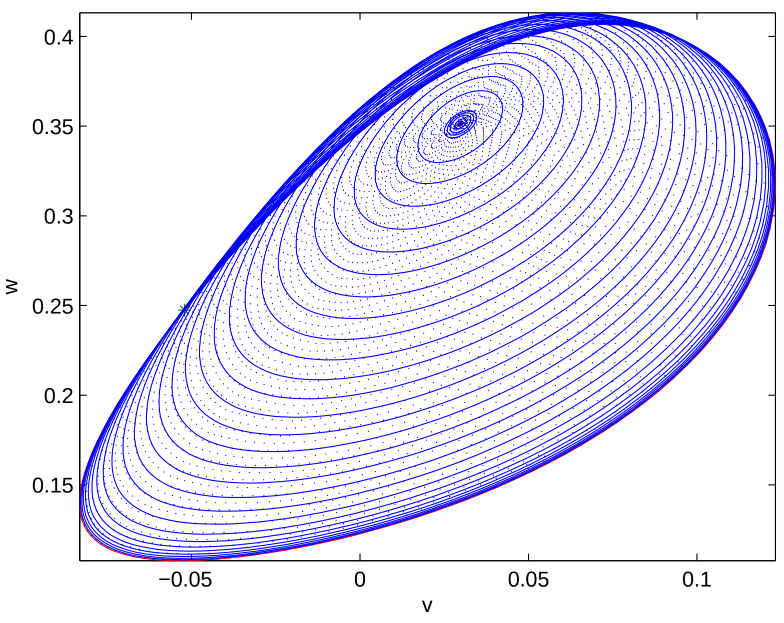


Figure 22: Computed fold of limit cycles curve started from a fold bifurcation of limitcycles

defining a NS point using a minimally extended system is

$$\left\{\begin{array}{ll} \frac{dx}{dt} - Tf(x,\alpha) & = 0 \\ x(0) - x(1) & = 0 \\ \int_0^1 \langle x(t), \dot{x}_{old}(t)\rangle dt & = 0 \\ G[x,T,\alpha] & = 0 \end{array}\right. \tag{78}$$

where

$$G = \begin{pmatrix} G^{11} & G^{12} \\ G^{21} & G^{22} \end{pmatrix}$$

is defined by requiring

$$N^3 \begin{pmatrix} v^1 & v^2 \\ G^{11} & G^{12} \\ G^{21} & G^{22} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{79}$$

Here v^1 and v^2 are functions and G^{11}, G^{12}, G^{21} and G^{22} are scalars and

$$N^3 = \begin{bmatrix} D - Tf_x(x(\cdot),\alpha) & w_{11} & w_{12} \\ \delta_0 - 2\kappa\delta_1 + \delta_2 & w_{21} & w_{22} \\ Int_{v_{01}} & 0 & 0 \\ Int_{v_{02}} & 0 & 0 \end{bmatrix} \tag{80}$$

where the bordering functions $v_{01}, v_{02}, w_{11}, w_{12}$, vectors $w_{21}$and w_{22} are chosen so that N^3 is nonsingular [12]. This method (using system (79) and (80)) is implemented in the curve definition file `neimarksacker`. The discretization is done using orthogonal collocation over the interval [0 2].

10.5.2 Bifurcations

In continuous-time systems there are eight generic codim 2 bifurcations that can be detected along the torus curve:

- *1:1 resonance.* We will denote this bifurcation by `R1`
- *2:1 resonance* point, denoted by `R2`
- *3:1 resonance* point, denoted by `R3`
- *4:1 resonance* point, denoted by `R4`
- *Fold-Neimarksacker* point, denoted by `LPNS`
- *Chenciner* point, denoted by `CH`.
- *Flip-Neimarksacker* point, denoted by `PDNS`
- *Double Neimarksacker bifurcation* point, denoted by `NSNS`

To detect these singularities, we first define 6 test functions:

- $\psi_1 = \kappa - 1$ (cf. formula (39))
- $\psi_2 = \kappa + 1$
- $\psi_3 = \kappa - 1/2$
- $\psi_4 = \kappa$
- $\psi_5 = (v_1^*)_{W_1}^T L_{C\times M} v_{1M}$
- $\psi_6 = Re(d)$
- $\psi_7 = det\,(M + I_n)$
- $\psi_8 = det\,((M2 \odot M2) - I_n)$

where v_{1M} is computed by solving

$$\left[\begin{array}{c} D - TA(t) + i\theta I \\ \delta_0 - \delta_1 \end{array}\right]_D v_{1M} = 0. \tag{81}$$

The normalization of v_{1M} is done by requiring $\sum_{i=1}^{N}\sum_{j=0}^{m}\sigma_j\langle(v_{1M})_{ij},(v_{1M})_{ij}\rangle t_i = 1$ where σ_j is the Gauss-Lagrange quadrature coefficient. By discretization we obtain

$$(v_{1W}^*)^H \left[\begin{array}{c} D - TA(t) + i\theta \\ \delta_0 + \delta_1 \end{array}\right]_{disc} = 0.$$

To normalize $(v_1^*)_{W_1}$ we require $\sum_{i=1}^{N}\sum_{j=1}^{m}|((v_1^*)_{W_1})_{ij}|_1 = 1$. Then $\int_0^1\langle v_1^*(t), v_1(t)\rangle dt$ is approximated by $(v_1^*)_{W_1}^T L_{C\times M} v_{1M}$ and if this quantity is nonzero, v_{1W}^* is rescaled so that $\int_0^1\langle v_1^*(t), v_1(t)\rangle dt = 1$. We compute φ_{1W}^* by solving

$$(\varphi_{1W}^*)^T \left[\begin{array}{c} D - TA(t) \\ \delta_0 - \delta_1 \end{array}\right]_{disc} = 0$$

and normalize $\varphi^*_{1W_1}$ by requiring $\sum_{i=1}^N \sum_{j=1}^m |((\varphi^*_1)_{W_1})_{ij}|_1 = 1$. Then $\int_0^1 \langle \varphi^*_1(t), F(u_{0,1}(t))\rangle dt$ is approximated by $(\varphi^*_1)^T_{W_1}(F(u_{0,1}(t)))_C$ and if this quantity is nonzero, φ^*_{1W} is rescaled so that $\int_0^1 \langle \varphi^*_1(t), F(u_{0,1}(t))\rangle dt = 1$. We compute $h_{20,1M}$ by solving

$$\begin{bmatrix} D - TA(t) + 2i\theta I \\ \delta_0 - \delta_1 \end{bmatrix}_{disc} h_{20,1M} = \begin{bmatrix} B(t, v_{1M}(t), v_{1M}(t)) \\ 0 \end{bmatrix}.$$

a_1 can be computed as $(\varphi^*_{W_1})^T(B(t, v_{1M}, \overline{v}_{1M}))_C$.

The computation of $(h_{11,1})_M$ is done by solving

$$\begin{bmatrix} (D - TA(t))_{C\times M} \\ \delta(0) - \delta(1) \\ (\varphi^*_{W_1})^T L_{C\times M} \end{bmatrix} (h_{11,1})_M = \begin{bmatrix} B(t; v_{1M}, \overline{v}_{1M}))_C - a_1(F(u_{0,1}(t)))_C \\ 0 \\ 0 \end{bmatrix}$$

The expression for the normal form coefficient d becomes

$$d = \tfrac{1}{2}((v^*_{1W_1})^T, (B(t; h_{11,1M}, v_{1M}))_C + (B(t; h_{20,1M}, \overline{v}_{1M}))_C + \tfrac{1}{T}(C(t; v_{1M}, v_{1M}, \overline{v}_{1M}))_C) \\ -\tfrac{a_1}{T}(v^*_{1W_1})^T(A(t)v_1(t))_C + \tfrac{ia_1\theta}{T^2}.$$

In the 7th test function, M is the monodromy matrix.

In the 8th test function, $M2 = (M - I_n)^2$, restricted to the subspace without the two eigenvalues with smallest norm.

The singularity matrix is:

$$S = \begin{pmatrix} 0 & - & - & - & - & - \\ - & 0 & - & - & - & - \\ - & - & 0 & - & - & - \\ - & - & - & 0 & - & - \\ - & - & - & - & 0 & - \\ - & - & - & - & 1 & 0 \end{pmatrix}. \tag{82}$$

10.5.3 Torus bifurcation initialization

One way to start a continuation of torus bifurcations of cycles supported in the current version is to start it from a torus bifurcation point or neimarksacker point (NS) on a limit cycle curve. This can be done using the following statement: `[x0,v0]=init_NS_NS(@odefile, x, s, ap, ntst, ncol)`. `x` should be the `x` as returned by the previous limit cycle continuation. `s` is the special point structure of the detected torus bifurcation point on the limit cycle curve. `odefile` specifies the ode-file to be used. `ap` is the array containing the two active parameters and `ntst` and `ncol` are again the number of mesh and collocation points for the discretization.

10.5.4 Example

We consider the following model of an autonomous electronic circuit [15] where x, y and z are state variables and $\beta, \gamma, r, a_3, b_3, \nu$ are parameters :

$$\begin{cases} \dot{x} &= (-(\beta+\nu)*x + \beta*y - a_3*x^3 + b_3*(y-x)^3)/r \\ \dot{y} &= \beta*x - (\beta+\gamma)*y - z - b_3*(y-x)^3 \\ \dot{z} &= y \end{cases} \tag{83}$$

A torus bifurcation in this system is described in the manual of [11]. It is found by starting an equilibrium curve from the trival equilibrium point ($x = y = z = 0$) at $\beta = 0.5$, $\gamma = -0.6$, $r = 0.6$, $a_3 = 0.32858$, $b_3 = 0.93358$, $\nu = -0.9$. The free parameter is ν and the branch is the trivial one ($x = y = z = 0$). On this branch a Hopf bifurcation is detected at $\nu = -0.58934$. On the emerging branch of limit cycles a branch point of limit cycles is found; by continuing the newly found branch one detects a torus bifurcation of periodic orbits.

We proceed in a somewhat different way; to avoid the branch point of periodic orbits we start with a slightly perturbed system where the last equation of (83) is replaced by

$$\dot{z} = y + \epsilon$$

with initially $\epsilon = 0.001$. The trivial solution is replaced by $x = 0.00125$, $y = -0.001$, $z = 0.00052502$ and we again compute a branch of equilibria with free parameter ν. The starting vector `x0` and its tangent vector `v0` are calculated from the following equilibrium curve continuation.

```
>> p=[0.5;-0.6;0.6;0.32858;0.93358;-0.9;0.001];
>> [x0,v0]=init_EP_EP(@torBPC,[0.00125;-0.001;0.00052502],p,[6]);
>> opt=contset; opt= contset(opt,'Singularities',1);
>> opt=contset(opt,'MaxNumPoints',10);
>> [x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = H , x = ( 0.005604 -0.001000 0.002702 -0.589286 )
First Lyapunov coefficient = -4.548985e-001

elapsed time  = 0.1 secs
npoints curve = 10
```

A Hopf point is found for $\nu = -0.58928$ for the state values $x = 0.0056037$, $y = -0.001$, $z = 0.0027021$. We start a curve of periodic orbits from this Hopf point, using $N = 25$ test intervals and $m = 4$ collocation points and choosing ν as the free parameter. The results can be seen in Figure 23.

```
>> x1=x(1:3,s(2).index);p(6)= x(end,s(2).index);
>> [x0,v0]=init_H_LC(@torBPC,x1,p,[6],0.0001,25,4);
>> opt=contset; opt= contset(opt,'Singularities',1);
>> opt=contset(opt,'MaxNumPoints',50);
>> opt=contset(opt,'Multipliers',1);
>> [x,v,s,h,f]=cont(@limitcycle,x0,v0,opt);
first point found
tangent vector to first point found
Limit point cycle (period = 8.411855e+000, parameter = -5.844928e-001)
Normal form coefficient = 1.788080e-001
Neimark-Sacker (period = 8.861103e+000, parameter = -5.957506e-001)
Normal form coefficient = 2.674115e-003
Period Doubling (period = 9.256846e+000, parameter = -6.146817e-001)
Normal form coefficient = -6.068973e-003
```

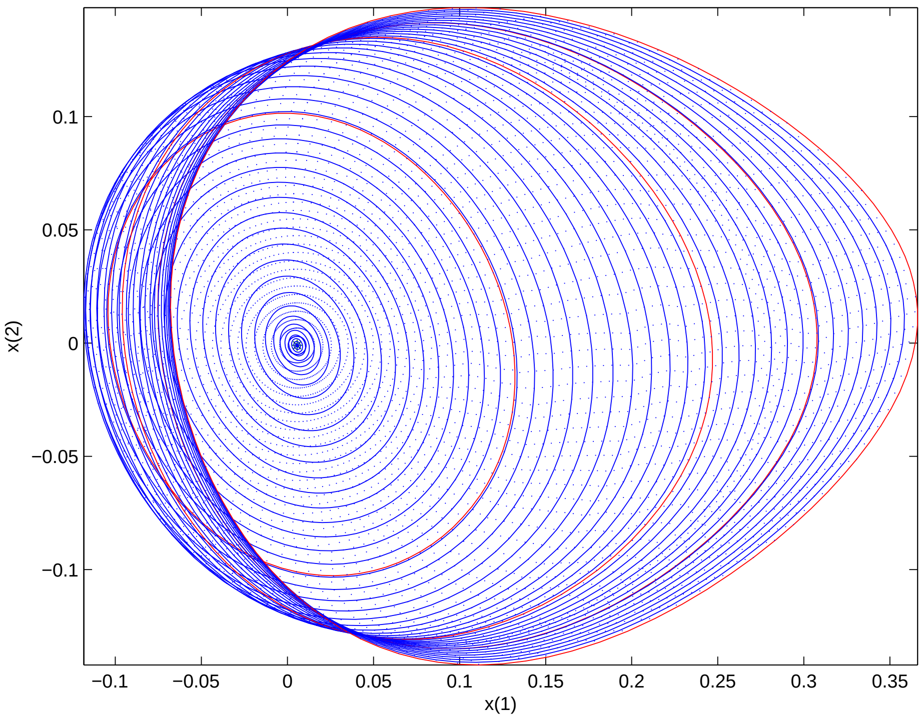


Figure 23: Computed limit cycle curve started from a Hopf bifurcation

```
elapsed time  = 28.3 secs
npoints curve = 50
>> plotcycle(x,v,s,[1 2]);
```

The previous computations are done by running the script testtorBPC1.

We detect a torus bifurcation point at $\nu = -0.59575$. To recover the torus bifurcation point of (83) we continue the torus bifurcation in two parameters β, ϵ. Choosing simply ϵ as a user functions we locate a NS bifurcation of (83). The starting vector `x0` is calculated from the NS on this branch using `init_NS_NS`. Continuation is done using a call to the standard continuer with `neimarksacker` as curve definition file.

```
>> [x1,v1]=init_NS_NS(@torBPC,x,s(3),[6 7],25,4);
>> opt=contset;opt = contset(opt,'VarTolerance',1e-4);
>> opt = contset(opt,'FunTolerance',1e-4);
>> opt=contset(opt,'Userfunctions',1);
>> UserInfo.name='epsilon0';UserInfo.state=1;UserInfo.label='E0';
>> opt=contset(opt,'UserfunctionsInfo',UserInfo);
>> opt=contset(opt,'Backward',1);opt=contset(opt,'MaxNumPoints',15);
>> [xns1,vns1,sns1,hns1,fns1]=cont(@neimarksacker,x1,v1,opt);
first point found
tangent vector to first point found
label = E0, x = ( 0.046835 0.141698 0.046209 ... )
elapsed time  = 30.2 secs
```

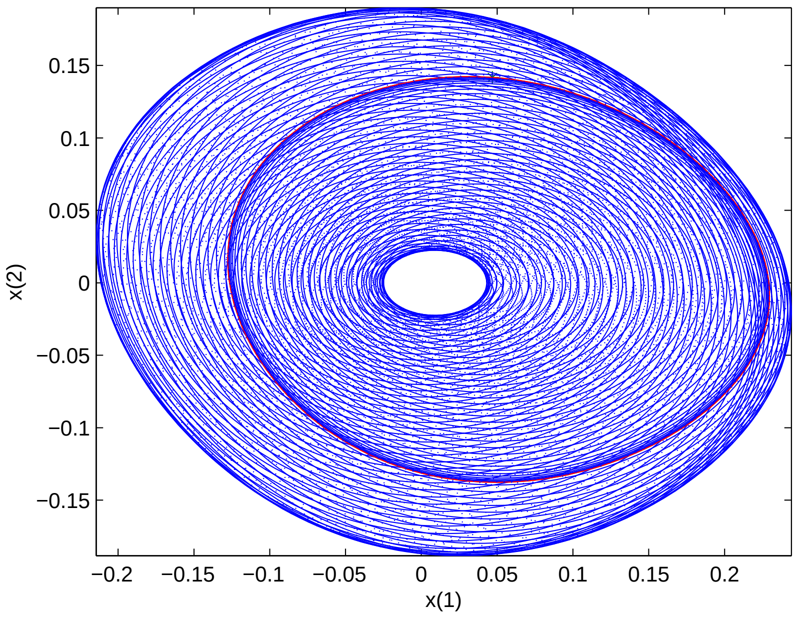


Figure 24: Computed torus of limit cycles curve started from a torus bifurcation of limit cycles

```
npoints curve = 15
```

This continuation is done by running the script testtorBPC2.

Finally, we continue numerically this NS orbit with two free parameters β, ν and find, interestingly, a closed curve of torus bifurcations of cycles. The results are plotted using the standard plot function `plotcycle` where the fourth argument is used to select the coordinates. A graphical representation of this phenomenon is shown in Figure 24. In the latter x is plotted versus y. The labels of the plot are added manually .

```
>> [x1,v1]=init_NS_NS(@torBPC,xns1,sns1(2),[1 6],50,4);
>> opt=contset;opt = contset(opt,'VarTolerance',1e-4);
>> opt = contset(opt,'FunTolerance',1e-4);
>> [xns2,vns2,sns2,hns2,fns2]=cont(@neimarksacker,x1,v1,opt);
first point found
tangent vector to first point found
Current step size too small (point 78)
elapsed time  = 826.2 secs
npoints curve = 78
>> plotcycle(xns2,vns2,sns2,[1 2]);
```

This continuation is done by running the script testtorBPC3.

11 Continuation of codim 2 bifurcations

11.1 Branch Point Continuation

11.1.1 Mathematical definition

In the toolbox branch point curves are computed by *minimally extended defining systems*, cf. [19], §4.1.2. The branch point curve is defined by the following system

$$\left\{ \begin{array}{rcl} f(u,\alpha) & = & 0, \\ g_1(u,\alpha) & = & 0, \\ g_2(u,\alpha) & = & 0, \end{array} \right. \tag{84}$$

where $(u,\alpha) \in \mathbf{R}^{n+2}$, while g_1 and g_2 are obtained by solving

$$N^4 \begin{pmatrix} v_{11} & v_{21} \\ v_{12} & v_{22} \\ g_1 & g_2 \end{pmatrix} = \begin{pmatrix} 0_n & 0_n \\ 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{85}$$

Here v_{11} and v_{21} are functions and v_{12}, v_{22}, g_1 and g_2 are scalars and

$$N^4 = \begin{bmatrix} f_u(u,\alpha) & f_\beta(u,\alpha) & w_{01} \\ v_0^{11T} & v_0^{12T} & 0 \\ v_0^{21T} & v_0^{22T} & 0 \end{bmatrix}$$

where the bordering functions $v_0^{11}, v_0^{21}, w_{01}$ and scalars v_0^{12}, v_0^{22} are chosen so that N^4 is nonsingular. This method is implemented in the curve definition file `branchpoint.m`.

11.1.2 Bifurcations

In the current version no bifurcations are detected.

11.1.3 Branch Point initialization

The only way to start a continuation of branch points supported in the current version is to start it from a Branch point detected on an equilibrium curve or on a fold curve. This can be done using the following statement: `[x0,v0]=init_BP_BP(@odefile, x, p, ap, bp)`. This routine stores its information in a global stucture `bpds`. The result of `init_BP_BP` contains a vector `x0` with the state variables and the three active parameters and a vector `v0` that is empty. Here `odefile` is the ode-file to be used, `x` is a vector of state variables containing the values of the state variables returned by a previous equilibrium curve or fold curve continuation, `p` is the vector containing the current values of the parameters, `ap` is the vector containing the indices of the 3 active parameters and `bp` is the index of the branch parameter.

11.1.4 Example

For this example the following system is used

$$\left\{ \; \dot{x} \;=\; \alpha_3 - (1+\lambda)x + \lambda\alpha_1/(1+\lambda\alpha_2 * e^{-\alpha_4 x/(1+x)}) \right. \tag{86}$$

where $\alpha_1 = 10 - 9 * \beta + \gamma, \alpha_2 = 10 - 9 * \beta$ and $\alpha_3 = -0.9 + 0.4 * \beta$. The model is coded in such a way that the parameters are $\lambda, \beta, \gamma, \alpha_4$, in that order.

It is easily seen that $x = -0.9$ is an equilibrium point of the system for the choice $(0, 0, 0, 3)$ of the parameters. From this we can start an equilbrium continuation with λ (the first parameter) free.

```
p=[0;0;0;3];ap1=[1];
[x0,v0]=init_EP_EP(@cstr,[-0.9],p,ap1);
opt=contset;
opt=contset(opt,'VarTolerance',1e-3);
opt=contset(opt,'FunTolerance',1e-3);
opt=contset(opt,'MaxNumPoints',50);
opt=contset(opt,'Singularities',1);
[x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = LP, x = ( -0.143564 1.250669 )
a=1.550147e+000
label = LP, x = ( 0.393180 0.377651 )
a=-7.370472e-001
cpl(x,v,s,[2,1]);
```

The results are plotted using the plot function `cpl` where the fourth argument is used to select the second and first components of the solution which are the parameter λ and the coordinate x. The resulting curve is a part of Figure 25. These computations can be done by running the script cstr1.

We start a fold continuation from the second LP detected on the previous equilibrium curve; λ and β are free in this run.

```
x1=x(1,s(3).index);
p(ap1)=x(end,s(3).index);
[x0,v0]=init_LP_LP(@cstr,x1,p,[1 2],[1 2 3 4]);
opt=contset(opt,'MaxNumPoints',300);
[x2,v2,s2,h2,f2]=cont(@limitpoint,x0,v0,opt);
first point found
tangent vector to first point found
label = BP1, x = ( 2.018621 0.581081 -4.709219 )
label = CP , x = ( 0.259553 1.968966 -0.090655 )
c=-8.847089e-001
label = BP1, x = ( 0.030643 1.772454 -0.127542 )
label = BP4, x = ( -0.000009 1.707401 -0.124964 )
label = CP , x = ( -0.173872 0.405524 0.608093 )
c=-2.263137e+000
label = BP4, x = ( -0.000000 0.421692 0.528995 )
Closed curve detected at step 164

elapsed time  = 0.5 secs
npoints curve = 164
```

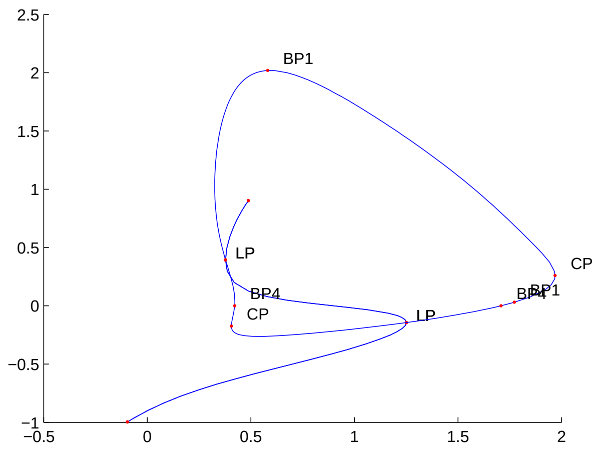


Figure 25: Computed fold curve

```
hold on;
cpl(x2,v2,s2,[2,1]);
```

These computations can be done by running the script cstr2. The results are plotted using the standard plot function `cpl` where the fourth argument is used to select the second and first components of the solution which are the parameter λ and the coordinate x. The results can be seen in Figure 25.

Finally, we continue numerically the BP curves with three free parameters λ, β and γ. The BP curves are started respectively from the first BP1 point (λ is the branch parameter) and the first BP4 point (α_4 is the branch parameter) detected on the previous fold curve. The results are plotted using the standard plot function `cpl` where the fourth argument is used to select the coordinates. A graphical representation of this phenomenon is shown in Figure 26. In the latter λ is plotted versus x. The labels of the plot are added manually .

```
x1=x2(1,s2(2).index);
p([1 2])=x2(end-1:end,s2(2).index);
[x0,v0]=init_BP_BP(@cstr,x1,p,[1 2 3],1);
opt=contset(opt,'Backward',1);
[x3,v3,s3,h3,f3]=cont(@branchpoint,x0,[],opt);
first point found
tangent vector to first point found

elapsed time  = 0.8 secs
npoints curve = 300
hold on;
cpl(x3,v3,s3,[2,1]);
x1=x2(1,s2(5).index);
p([1 2])=x2(end-1:end,s2(5).index);
[x0,v0]=init_BP_BP(@cstr,x1,p,[1 2 3],4);
opt=contset(opt,'Backward',1);
```

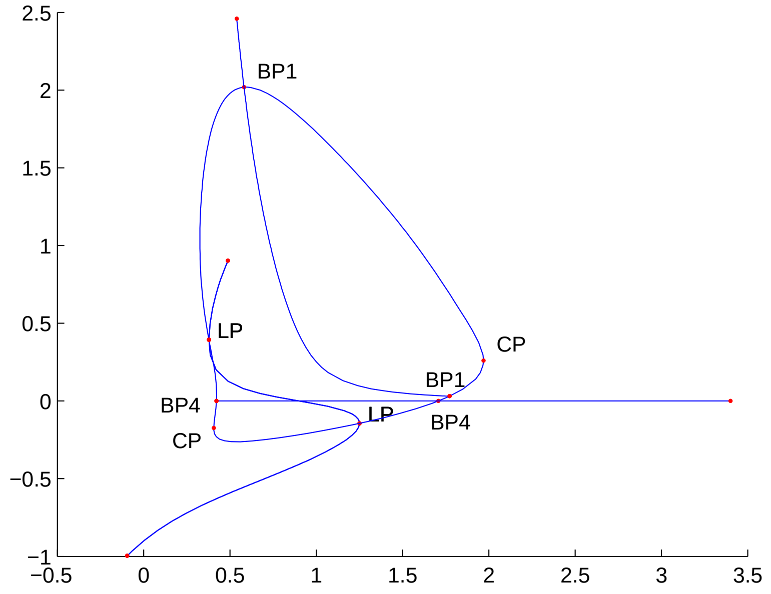


Figure 26: Computed BP curves started from Branch Points detected on a fold curve.

```
[x3,v3,s3,h3,f3]=cont(@branchpoint,x0,[],opt);
first point found
tangent vector to first point found

elapsed time  = 0.8 secs
npoints curve = 300
hold on;
cpl(x3,v3,s3,[2,1]);
```

These computations can be done by running the script cstr3.

11.2 Branch Point of Cycles Continuation

11.2.1 Mathematical Definition

A BPC can be characterized by adding two extra constraints $G_1 = 0$ and $G_2 = 0$ to (50) where G_1 and G_2 are the Branch Point test functions. The complete BVP defining a BPC point using the minimal extended system is

$$\left\{\begin{array}{ll} \frac{dx}{dt} - Tf(x,\alpha) & = 0 \\ x(0) - x(1) & = 0 \\ \int_0^1 \langle x(t), \dot{x}_{old}(t)\rangle dt & = 0 \\ G[x,T,\alpha] & = 0 \end{array}\right. \tag{87}$$

where

$$G = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix}$$

is defined by requiring

$$N \begin{pmatrix} v_1 & v_2 \\ G_1 & G_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{88}$$

Here v_1 and v_2 are functions, G_1 and G_2 are scalars and

$$N = \begin{bmatrix} D - Tf_x(x(\cdot),\alpha) & -f(x(\cdot),\alpha) & -Tf_\beta(x(\cdot),\alpha) & w_{01} \\ \delta_1 - \delta_0 & 0 & 0 & w_{02} \\ Int_{\dot{x}_{old}(\cdot)} & 0 & 0 & w_{03} \\ v_{11} & v_{12} & v_{13} & 0 \\ v_{21} & v_{22} & v_{23} & 0 \end{bmatrix} \tag{89}$$

where the bordering operators v_{11}, v_{21}, function w_{01}, vector w_{02} and scalars $v_{12}, v_{22}, v_{13}, v_{23}$ and w_{03} are chosen so that N is nonsingular [12][13].

11.2.2 Bifurcations

In the current version no bifurcations are detected.

11.2.3 Branch Point of Cycles initialization

The only way to start a continuation of branch points supported in the current version is to start it from a BPC detected on an limitcycle curve or on an LPC curve. This can be done using the following statement: `[x0,v0]=init_BPC_BPC(@odefile, x, s, ap, ntst, ncol, bp)`. This routine stores its information in a global stucture `lds`. The result of `init_BP_BP` contains a vector `x0` with the state variables and the three active parameters and a vector `v0` that is empty. Here `odefile` is the ode-file to be used, `x` is a vector of state variables containing the values of the state variables returned by a previous limit cycle curve or fold of cycles curve continuation, `s` is a structure containing the BPC point values returned by a previous limit cycle curve or fold of cycles curve continuation, `ap` is the vector containing the indices of the 3 active parameters and `bp` is the index of the branch parameter. `ntst` and `ncol` are again the number of mesh and collocation points for the discretization.

11.2.4 Example

In this section we discuss a non-generic situation, i.e. a case with a symmetry and a continuation of BPC points that involves two effective parameters and one artificial parameter.

For this example the following model is used:

$$\begin{cases} \dot{x} &= (-(\beta+\nu)x + \beta y - a_3x^3 + b_3(y-x)^3)/r \\ \dot{y} &= \beta x - (\beta+\gamma)y - z - b_3(y-x)^3 \\ \dot{z} &= y \end{cases} \tag{90}$$

which is the same system as in the torus of cycles continuation. It has a trivial solution branch $x = y = z = 0$ for all parameter values. Moreover, it has the Z_2 - symmetry $x \to -x, y \to -y$. To compute the branch of BPC points with respect to ν through the BPC point that we will detect on a limitcycle continued with free parameters ν, β, we need to introduce an additional free parameter that breaks the symmetry. There are many choices for this; we choose to introduce a parameter ϵ and extend the system (90) by simply adding a term $+\epsilon$ to the first right-hand-side. For $\epsilon = 0$ this reduces to (90) while for $\epsilon \neq 0$ the symmetry is broken.

We start by computing the trivial branch with fixed parameters $\gamma = -0.6$, $r = 0.6$, $a_3 = 0.328578$, $b_3 = 0.933578$, $\beta = 0.5$,$\epsilon = 0$ and free parameter ν with initially $\nu = -0.9$. On this branch a Hopf point is detected for $\nu = -0.58933644$ and a branch point of equilibria for $\nu = -0.5$.

```
>> p=[0.5;-0.6;0.6;0.32858;0.93358;-0.9;0];
>> [x0,v0]=init_EP_EP(@torBPC,[0;0;0],p,[6]);
>> opt=contset; opt= contset(opt,'Singularities',1);
>> opt=contset(opt,'MaxNumPoints',50);
>> [x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = H , x = ( 0.000000 0.000000 0.000000 -0.589336 )
First Lyapunov coefficient = -4.563631e-001
label = BP, x = ( 0.000000 0.000000 0.000000 -0.500000 )

elapsed time  = 0.3 secs
npoints curve = 50
```

These computations can be done by running the script testtorBPC4.

From the Hopf point we start the computation of a curve of limit cycles, using 25 test intervals and 4 collocation points. This is clearly a branch of symmetric solutions of (90); we detect one LPC and two BPC, see Fig. 27.

```
>> x1=x(1:3,s(2).index);p(6)= x(end,s(2).index);ap = 6;
>> [x0,v0]=init_H_LC(@torBPC,x1,p,ap,0.0001,25,4);
>> opt=contset; opt= contset(opt,'Singularities',1);
>> opt=contset(opt,'Multipliers',1);
>> opt=contset(opt,'MaxNumPoints',150);
>> opt=contset(opt,'Adapt',5);
>> [xlc,vlc,slc,hlc,flc]=cont(@limitcycle,x0,v0,opt);
first point found
tangent vector to first point found
Limit point cycle (period = 8.426472e+000, parameter = -5.843348e-001)
Normal form coefficient = 1.553595e-001
Branch Point cycle(period = 8.689669e+000, parameter = -5.870290e-001)
Neimark-Sacker (period = 8.743033e+000, parameter = -5.881194e-001)
Neutral saddle

elapsed time  = 30.2 secs
npoints curve = 150
>> plotcycle(xlc,vlc,slc,[1 2]);
```

These computations can be done by running the script testtorBPC5.

We continue the secondary cycle branch passing through the BPC point. From Fig. 28 it is clear that in the secondary cycle the symmetry is broken.

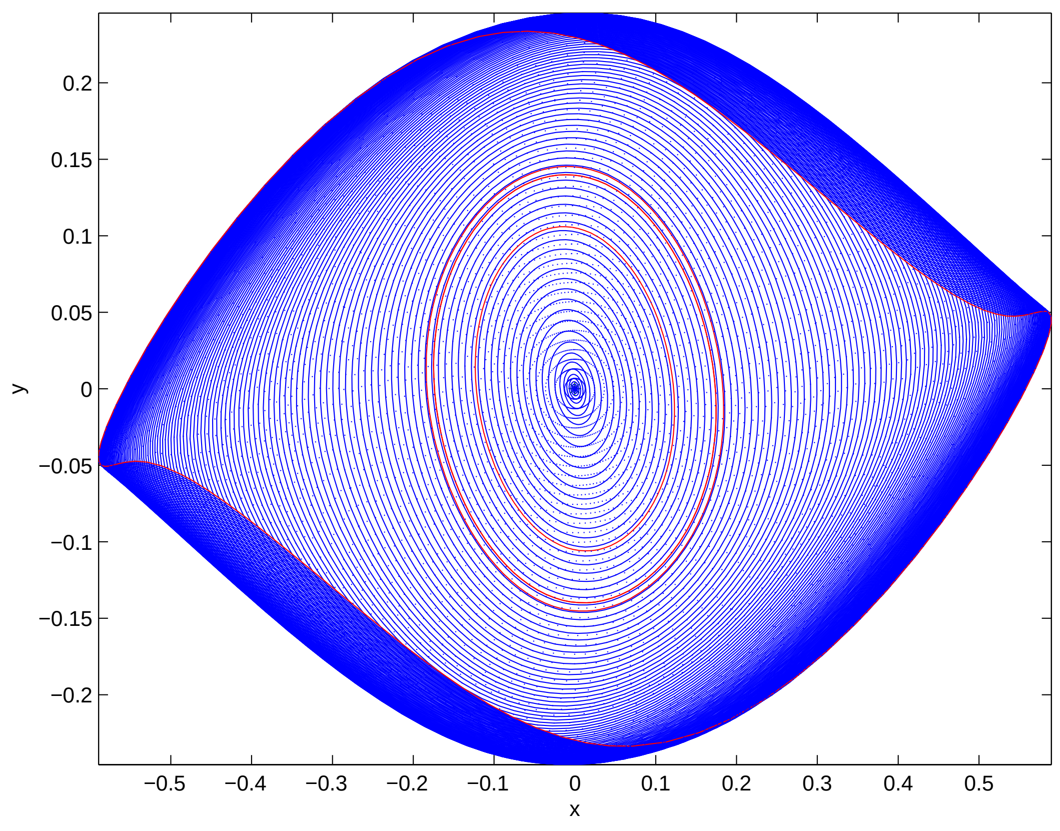


Figure 27: Curve of limit cycles with LPC and branch points in the circuit example.

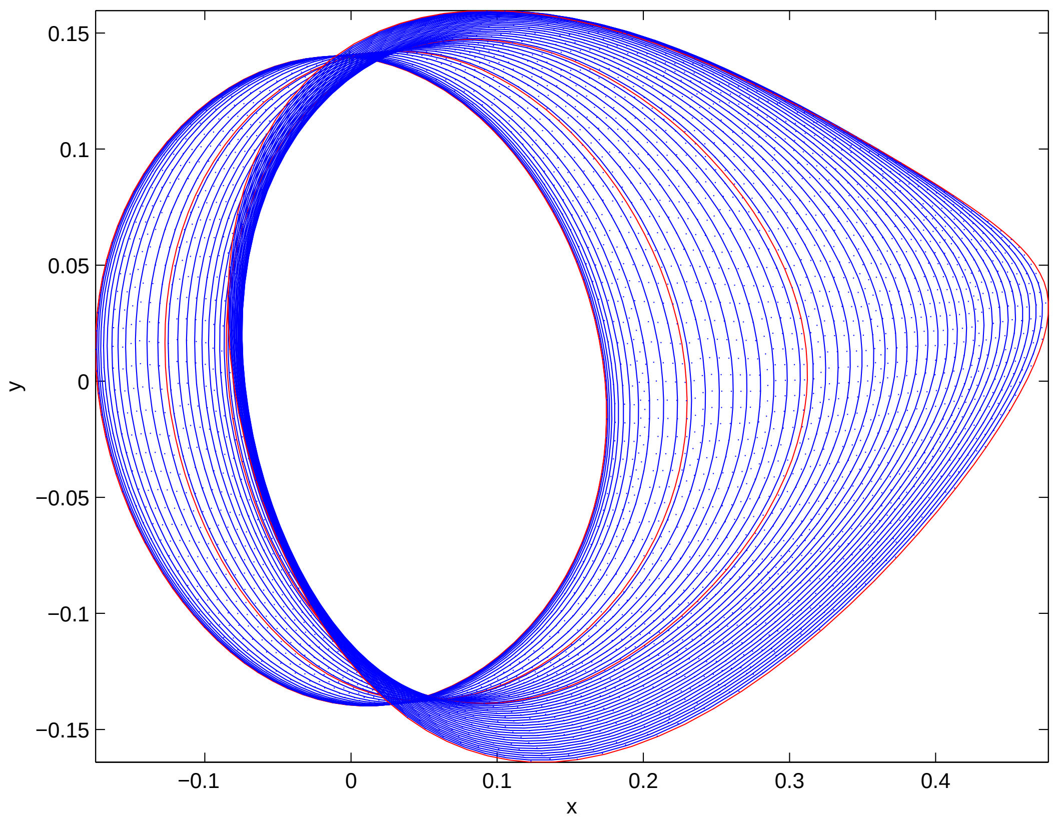


Figure 28: Asymetric curve of limit cycles in the circuit example.

```
>> [x1,v1]=init_BPC_LC(@torBPC,xlc,vlc,slc(3),25,4,1e-6);
>> opt=contset(opt,'MaxNumPoints',50);
>> opt=contset(opt,'Backward',1);
>> [xlc1,vlc1,slc1,hlc1,flc1]=cont(@limitcycle,x1,v1,opt);
first point found
tangent vector to first point found
Neimark-Sacker (period = 8.794152e+000, parameter = -5.916502e-001)
Normal form coefficient = -8.661266e-003
Period Doubling (period = 9.266303e+000, parameter = -6.149552e-001)
Normal form coefficient = -6.374237e-003

elapsed time  = 13.6 secs
npoints curve = 50
>> plotcycle(xlc1,vlc1,slc1,[1 2]);
```

These computations can be done by running the script testtorBPC6.

Using the code for the continuation of generic BPC points with three free parameters ν, β, ϵ we continue the curve of non-generic BPC points, where ϵ remains close to zero. The picture in Fig. 29 clearly shows that the symmetry is preserved.

```
>> [x1,v1]=init_BPC_BPC(@torBPC,xlc,slc(3),[1 6 7],25,4,ap);
```

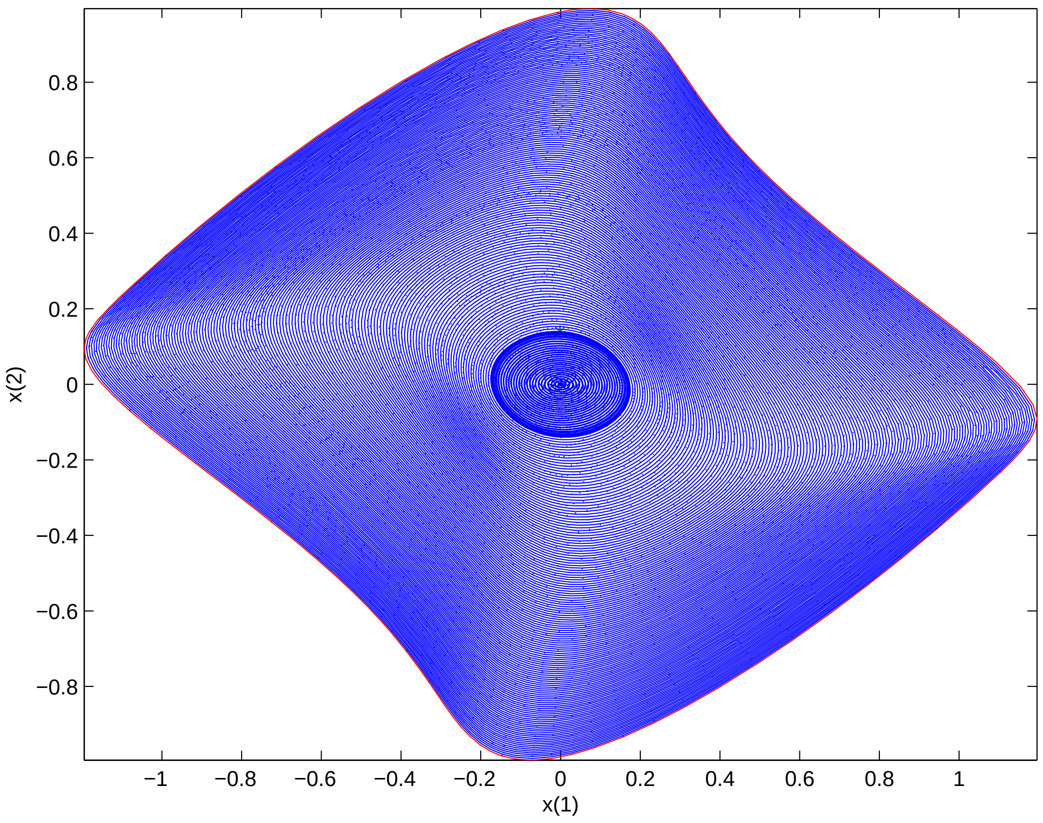


Figure 29: Curve of BPC points in the circuit example.

```
>> opt=contset(opt,'MaxNumPoints',200);
>> [xbpc,vbpc,sbpc,hbpc,fbpc]=cont(@branchpointcycle,x1,v1,opt);
first point found
tangent vector to first point found

elapsed time  = 158.1 secs
npoints curve = 200
>> plotcycle(xbpc,vbpc,sbpc,[1 2]);
```

These computations can be done by running the script testtorBPC7.

12 Continuation of homoclinic orbits

12.1 Mathematical definition

In dynamical systems theory, an orbit corresponding to a solution $u(t)$ is called homoclinic to the equilibrium point u^0 of the dynamical system if $u(t) \to u^0$ as $t \to \pm\infty$. There are two types of homoclinic orbits with codimension 1, namely homoclinic-to-hyperbolic-saddle (HHS), if u^0 is a saddle (saddle-focus or bi-focus), and homoclinic-to-saddle-node (HSN), if u^0 is a saddle-node (i.e., exhibits a limit point bifurcation). We recall that AUTO has a toolbox for homoclinic continuation, named HomCont [4], [5].

During the continuation, it is necessary to keep track of several eigenspaces of the equilibrium in each step. To do this in an efficient way, MatCont incorporates the continuation of invariant subspaces [7] into the defining system. For some details on the implementation of the homoclinic continuation we refer to [16].

12.1.1 Homoclinic-to-Hyperbolic-Saddle Orbits

To continue HHS orbits in two free parameters, we use an extended defining system that consists of several parts.

First, the infinite time interval is truncated, so that instead of $[-\infty, +\infty]$ we use $[-T, +T]$, which is scaled to $[0, 1]$ and divided into mesh-intervals. The mesh is nonuniform and adaptive. Each mesh interval is further subdivided by equidistant fine mesh points. Also, each mesh interval contains a number of collocation points. (This discretization is the same as that in AUTO for boundary value problems.) The equation

$$\dot{x}(t) - 2Tf(x(t), \alpha) = 0, \tag{91}$$

must be satisfied in each collocation point.

The second part is the equilibrium condition

$$f(x_0, \alpha) = 0. \tag{92}$$

Third, there is a so-called phase condition needed for the homoclinic solution, similar to periodic solutions

$$\int_0^1 \dot{\widetilde{x}}^*(t)[x(t) - \widetilde{x}(t)]dt = 0. \tag{93}$$

Here $\widetilde{x}(t)$ is some initial guess for the solution, typically obtained from the previous continuation step. We note that in the literature another phase condition is also used, see, for example [10]. However, in the present implementation we employ the condition (93).

Fourth, there are the homoclinic-specific constraints to the solution. For these we need access to the stable and unstable eigenspaces of the system in the equilibrium point after each step. It is not efficient to recompute the spaces from scratch in each continuation-step. Instead, we use the algorithm for continuing invariant subspaces, as described in [7]. This method adds two small-sized vectors (Y_S and Y_U) to the system variables, from which the necessary eigenspaces (stable and unstable, respectively) can easily be computed in each step.

If $Q(0)$ is an orthogonal matrix whose first m columns form a basis for the invariant subspace under consideration in the previous step, and $A = f_x(x_0, \alpha)$ is the Jacobian at the new equilibrium point, then we first compute the so-called Ricatti-blocks, T_{ij}, by the formula

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} = Q(0)^T A\, Q(0). \tag{94}$$

If n is the number of state variables, then T_{11} is of size $m \times m$ and T_{22} is $(n-m) \times (n-m)$. This is done for the stable and unstable eigenspaces separately. Now Y_S and Y_U are obtained from the Ricatti equations

$$\begin{array}{lllllllll} T_{22U}Y_U & - & Y_U T_{11U} & + & T_{21U} & - & Y_U T_{12U} Y_U & = & 0, \\ T_{22S}Y_S & - & Y_S T_{11S} & + & T_{21S} & - & Y_S T_{12S} Y_S & = & 0. \end{array} \tag{95}$$

Now we can formulate constraints on the behavior of the solution close to the equilibrium x_0. The initial vector of the orbit, $(x(0) - x_0)$, is placed in the unstable eigenspace of the system in the equilibrium. We express that by the requirement that it is orthogonal to the orthogonal complement of the unstable eigenspace. Using Y_U, we can compute the orthogonal complement of the unstable eigenspace. If $Q_U(0)$ is the orthogonal matrix from the previous step, related to the unstable invariant subspace, then a basis for the orthogonal complement in the new step $Q_U^{\perp}(s)$ is

$$Q_U^{\perp}(s) = Q_U(0) \begin{bmatrix} -Y_U^T \\ I \end{bmatrix}.$$

Note that $Q_U^{\perp}(s)$ is not orthogonal. The full orthogonal matrix Q_{1U} needed for the next step, is computed separately after each step. The equations to be added to the system are (after analogous preparatory computations for the stable eigenspace)

$$\begin{array}{l} Q_U^{\perp}(s)^T(x(0) - x_0) = 0, \\ Q_S^{\perp}(s)^T(x(1) - x_0) = 0. \end{array} \tag{96}$$

Finally, the distances between $x(0)$ (resp., $x(1)$) and x_0 must be small enough, so that

$$\begin{array}{l} \|x(0) - x_0\| - \epsilon_0 = 0, \\ \|x(1) - x_0\| - \epsilon_1 = 0. \end{array} \tag{97}$$

A system consisting of all equations (91), (92), (93), (95), (96) and (97), is overdetermined. The basic defining system for the continuation of a HHS orbit in two free parameters consists of (91), (92), (95), (96), and (97) with fixed $\epsilon_{0,1}$, so that the phase condition (93) is not used. The variables in this system are stored in one vector. It contains the values of $x(t)$ in the fine mesh points including $x(0)$ and $x(1)$, the truncation time T, two free system parameters, the coordinates of the saddle x_0, and the elements of the matrices Y_S and Y_U. Alternatively, the phase condition (93) can be added if T is kept fixed but ϵ_0 and ϵ_1 are allowed to vary. It is also possible to fix T and ϵ_0, say, and allow ϵ_1 to vary, again with no phase condition. Other combinations are also possible, in particular, when the homotopy method [7] is used to compute a starting homoclinic solution.

12.1.2 Homoclinic-to-Saddle-Node Orbits

For a homoclinic orbit to a saddle-node equilibrium, the extended defining system undergoes some small changes. Now $(x(0) - x_0)$ has to be placed in the center-unstable subspace. Analogously, $(x(1) - x_0)$ must be in the center-stable subspace. This again is implemented by requiring that the vector is orthogonal to the orthogonal complement of the corresponding space. So the equations (96) themselves do not really change; the changes happen in the computation of the matrices Q. The defining system now has one equation less than in the HHS case ($n_s + n_u < n$, with n_s the dimension of the stable, and n_u of the unstable eigenspace); the number of equations is restored however, by adding the constraint that the equilibrium must be a saddle-node. For this we use the bordering technique, as described in section 4.2.1 of [19].

12.2 Bifurcations

During HSN continuation, only one bifurcation is tested for, namely the non-central homoclinic-to-saddle-node orbit or NCH. This orbit forms the transition between HHS and HSN curves. The strategy used for detection is taken from HomCont [5].

During HHS continuation, all bifurcations detected in HomCont are also detected in our implementation. For this, mostly test functions from [5] are used.

Suppose that the eigenvalues of $f_x(x_0, \alpha_0)$ can be ordered according to

$$\Re(\mu_{ns}) \leq ... \leq \Re(\mu_1) < 0 < \Re(\lambda_1) \leq ... \leq \Re(\lambda_{nu}), \tag{98}$$

where $\Re()$ stands for 'real part of', ns is the number of stable, and nu the number of unstable eigenvalues. The test functions for the bifurcations are

- Neutral saddle, saddle-focus or bi-focus

  $$\psi = \Re(\mu_1) + \Re(\lambda_1)$$

  If both μ_1 and λ_1 are real, then it is a neutral saddle, if one is real and one consists of a pair of complex conjugates, the bifurcation is a saddle-focus, and it is a bi-focus when both eigenvalues consist of a pair of complex conjugates.

- Double real stable leading eigenvalue

  $$\psi = \begin{cases} (\Re(\mu_1) - \Re(\mu_2))^2, & \Im(\mu_1) = 0 \\ -(\Im(\mu_1) - \Im(\mu_2))^2, & \Im(\mu_1) \neq 0 \end{cases}$$

- Double real unstable leading eigenvalue

  $$\psi = \begin{cases} (\Re(\lambda_1) - \Re(\lambda_2))^2, & \Im(\lambda_1) = 0 \\ -(\Im(\lambda_1) - \Im(\lambda_2))^2, & \Im(\lambda_1) \neq 0 \end{cases}$$

- Neutrally-divergent saddle-focus (stable)

  $$\psi = \Re(\mu_1) + \Re(\mu_2) + \Re(\lambda_1)$$

- Neutrally-divergent saddle-focus (unstable)

$$\psi = \Re(\mu_1) + \Re(\lambda_2) + \Re(\lambda_1)$$

- Three leading eigenvalues (stable)

$$\psi = \Re(\mu_1) - \Re(\mu_3)$$

- Three leading eigenvalues (unstable)

$$\psi = \Re(\lambda_1) + \Re(\lambda_3)$$

- Non-central homoclinic-to-saddle-node

$$\psi = \Re(\mu_1)$$

- Shil'nikov-Hopf

$$\psi = \Re(\lambda_1)$$

- Bogdanov-Takens point

$$\psi = \begin{cases} \Re(\mu_1) \\ \Re(\lambda_1) \end{cases}$$

For orbit- and inclination-flip bifurcations, we assume the same ordering of the eigenvalues of $f_x(x_0, \alpha_0) = A(x_0, \alpha_0)$ as shown in (98), but also that the leading eigenvalues μ_1 and λ_1 are unique and real:

$$\Re(\mu_{ns}) \leq ... \leq \Re(\mu_2) < \mu_1 < 0 < \lambda_1 < \Re(\lambda_2) \leq ... \leq \Re(\lambda_{nu}) \ .$$

Then it is possible to choose normalised eigenvectors p_1^s and p_1^u of $A^T(x_0, \alpha_0)$ and q_1^s and q_1^u of $A(x_0, \alpha_0)$ depending smoothly on (x_0, α_0), which satisfy

$$\begin{aligned} A^T(x_0, \alpha_0)\ p_1^s &= \mu_1\ p_1^s \qquad & A^T(x_0, \alpha_0)\ p_1^u &= \lambda_1\ p_1^u \\ A(x_0, \alpha_0)\ q_1^s &= \mu_1\ q_1^s \qquad & A(x_0, \alpha_0)\ q_1^u &= \lambda_1\ q_1^u \ . \end{aligned}$$

The test functions for the orbit-flip bifurcations are then:

- Orbit-flip with respect to the stable manifold

$$\psi = e^{-\mu_1 T} < p_1^s, x(1) - x_0 >$$

- Orbit-flip with respect to the unstable manifold

$$\psi = e^{\lambda_1 T} < p_1^u, x(0) - x_0 >$$

For the inclination-flip bifurcations, in [5] the following test functions are introduced:

- Inclination-flip with respect to the stable manifold

$$\psi = e^{-\mu_1 T} < q_1^s, \phi(0) >$$

- Inclination-flip with respect to the unstable manifold

$$\psi = e^{\lambda_1 T} < q_1^u, \phi(1) >$$

where ϕ ($\phi \in \mathcal{C}^1([0,1], \mathbb{R}^n)$) is the solution to the adjoint system, which can be written as

$$\begin{cases} \dot{\phi}(t) + 2\,T\,A^T(x(t), \alpha_0)\,\phi(t) = 0 \\ (L_s)^T \phi(1) = 0 \\ (L_u)^T \phi(0) = 0 \\ \int_0^1 \widetilde{\phi}^T(t)[\phi(t) - \widetilde{\phi}(t)]dt = 0 \end{cases} \tag{99}$$

where L_s and L_u are matrices whose columns form bases for the stable and unstable eigenspaces of $A(x_0, \alpha_0)$, respectively, and the last condition selects one solution out of the family $c\phi(t)$ for $c \in \mathbb{R}$. L_u is equivalent to Q_U from the mathematical definition of the system, and L_s to Q_S. In the homoclinic defining system the orthogonal complements of Q_S and Q_U are used; in the adjoint system for the inclination-flip bifurcation, we use the matrices themselves (or at least, their transposed versions).

12.3 Homoclinic initialization

For homoclinics the same problems occur as with limit cycles, a homoclinic continuation can't be done by just calling the continuer as

```
[x,v,s,h,f]=cont(@homoclinic, x0, v0, opt)
```

The homoclinic curve file has to know

- which ode file to use,
- which parameter is active,
- the values of all parameters,
- the number of mesh and collocation points to use for the discretization.

Also an initial cycle `x0` has to be known. All this information can be supplied using an initializer. All initializers return an initial cycle `x0` as well as its tangent vector `v0`.

- `[x0,v0]=init_LC_Hom(@odefile, x, s, p, ap, ntst, ncol, extravec, T, eps0, eps1)`
  Calculates an initial homoclinic orbit from a limit cycle with large period. Here `odefile` is the ode-file to be used. `x` and `s` are here the `x` and `s` belonging to the limit cycle with large period, obtained in a previous continuation. `p` is the vector containing the current values of the parameters. `ap` is the active parameter and `ntst` and `ncol` are the number of mesh and collocation points to be used for the discretization. `extravec` is a vector of 3 integers, which are either `0` or `1`, and which indicate which of `T, eps0, eps1` are to be variable during the continuation. The vector is `1` for those that should be variable. This can either be 1 or 2 or the three parameters. `T, eps0` and `eps1` are values for these parameters.

  In the GUI-version MatCont, you call do this by selecting the cycle with large period as initial point, and then defining it (Type → Initial Point) as a Homoclinic-to-Saddle.

- `[x0,v0]=init_BT_Hom(@odefile, x, s, p, ap, ntst, ncol, eps, extravec)`
  Calculates an initial homoclinic orbit from a Bogdanov-Takens bifurcation, detected on a Hopf or Limit Point curve. All parameters are similar to the ones for initialisation from a limit cycle, except for `eps`: it is the amplitude of the initial homoclinic orbit, comparable to `h` for the initialisation of a limit cycle from a Hopf point.

- `[x0,v0]=init_Hom_Hom(@odefile, x, s, p, ap, ntst, ncol, extravec, T, eps0, eps1)`
  Calculates an initial homoclinic orbit from a homoclinic orbit obtained during a previous continuation. All parameters are similar to the initialisations above.

For homoclinic-to-saddle-node orbits, the user needs to replace `Hom` with `HSN` in the above commands. Note that there are no HSN orbits emanating from a BT point, so only the initializers from a limit cycle and a previously computed HSN are available.

12.4 Examples

12.4.1 CL_MatCont: the MLFast example

In the experiment for continuing LPCs in this manual, we already mentioned that the limit cycles were approaching a homoclinic orbit. We will now approach this homoclinic even closer, and then start their continuation from the large limit cycle. The result is shown in Figure 30.

```
>> init;
>> p=[0.11047;0.1];ap1=[1];
>> [x0,v0]=init_EP_EP(@MLfast,[0.047222;0.32564],p,ap1);
>> opt=contset;opt=contset(opt,'Singularities',1);
>> opt=contset(opt,'MaxNumPoints',65);
>> opt=contset(opt,'MinStepSize',0.00001);
>> opt=contset(opt,'MaxStepSize',0.01);
>> opt=contset(opt,'Backward',1);
>> [x,v,s,h,f]=cont(@equilibrium,x0,[],opt);
first point found
tangent vector to first point found
label = H , x = ( 0.036756 0.294770 0.075659 )
First Lyapunov coefficient = 8.234573e+000
label = LP, x = ( -0.033738 0.136501 -0.020727 )
a=-1.036706e+001
label = H , x = ( -0.119894 0.045956 0.033207 )
Neutral saddle
label = LP, x = ( -0.244915 0.008514 0.083257 )
a=2.697414e+000

elapsed time  = 0.4 secs
npoints curve = 65
>> x1=x(1:2,s(2).index);p=[x(end,s(2).index);0.1];
>> [x0,v0]=init_H_LC(@MLfast,x1,p,ap1,0.0001,30,4);
>> opt=contset;
>> opt=contset(opt,'MaxStepSize',1);
>> opt=contset(opt,'IgnoreSingularity',1);
>> opt=contset(opt,'Singularities',1);
>> opt=contset(opt,'MaxNumPoints',200);
>> [x2,v2,s2,h2,f2]=cont(@limitcycle,x0,v0,opt);
first point found
tangent vector to first point found
Limit point cycle (period = 4.222011e+000, parameter = 8.456948e-002)
Normal form coefficient = -2.334576e-001
Limit point cycle (period = 5.653399e+001, parameter = 7.293070e-002)
Normal form coefficient = 1.132235e+000
Limit point cycle (period = 5.739877e+001, parameter = 7.293070e-002)
Normal form coefficient = 3.266287e+000
Limit point cycle (period = 8.938964e+001, parameter = 7.293071e-002)
Normal form coefficient = -1.537206e-001

elapsed time  = 86.6 secs
npoints curve = 200
>> p(ap1) = x2(end,end);
>> T = x2(end-1,end)/2;
>> [x0,v0]=init_LC_Hom(@MLfast, x2(:,end), s2(:,end), p, [1 2], 40, 4,...
>>          [0 1 1], T, 0.01, 0.01);
>> opt=contset(opt,'MaxNumPoints',15);
>> [xh,vh,sh,hh,fh] = cont(@homoclinic,x0,v0,opt);
first point found
tangent vector to first point found
elapsed time  = 4.4 secs
npoints curve = 15
>> plotcycle(xh,vh,sh,[1 2]);
```

The above computations can be done by running the script `homocl`. The picture is presented in Figure 30.

12.4.2 MatCont: the Koper example

Consider the following system of differential equations:

$$\left\{\begin{array}{rcl} \epsilon_1\dot{x} &=& (k\,y - x^3 + 3x - \lambda) \\ \dot{y} &=& x - 2y + z \\ \dot{z} &=& \epsilon_2(y - z). \end{array}\right. \tag{100}$$

This system, which is a three-dimensional van der Pol-Duffing oscillator, has been introduced and studied by [22]. It is used as a standard demo in HomCont. Parameters ϵ_1 and ϵ_2 are kept at 0.1 and 1, respectively. We note that system (100) has a certain symmetry: If $(x(t), y(t), z(t))$ is a solution for a given value of λ, then $(-x(t), -y(t), -z(t))$ is a solution for $-\lambda$.

Starting from a general point $(0, -1, 0.1)$ and setting $k = 0.15$ and $\lambda = 0$, we find by time integration a stable equilibrium at $(-1.775, -1.775, -1.775)$.

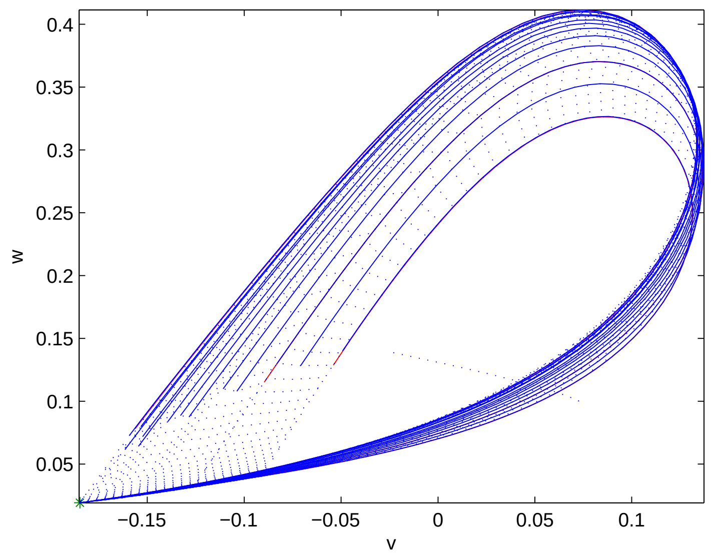


Figure 30: Computed curve of homoclinic-to-saddle orbits started from a limit cycle with large period.

By equilibrium continuation with λ free, we find two limit points (LP), at (−1.024695, −1.024695, −1.024695) for $\lambda = -2.151860$ with normal form coefficient $a = -4.437056e+000$ and at (1.024695, 1.024695, 1.024695) for $\lambda = 2.151860$ with normal form coefficient $a = 4.437060e+000$ (note the reflection).

By continuation of the limit points with (k, λ) free, MatCont detects a cusp point CP at $(0,0,0)$ for $k = -3$ and $\lambda = 0$. Also detected are two Zero-Hopf points ZH for $k = -0.3$ at ±(0.948683, 0.948683, 0.948683) and $\lambda = \pm1.707630$, but these are in fact Neutral Saddles. Further, two Bogdanov-Takens points BT are found for $k = -0.05$ at ±(0.991632, 0.991632, 0.991632) and $\lambda = \pm1.950209$. The normal form coefficients are $(a, b) = (6.870226e+000, 3.572517e+001)$.

A bifurcation diagram of (100) is shown in Figure 31.

We will now start an orbit homoclinic to saddle from a limit cycle with large period. First select the BT point at $\lambda = 1.950209$, and then compute a curve of Hopf points H passing through it (Type → Curve → Hopf), along which one encounters a Generalized Hopf bifurcation GH where $\lambda = 1.09180$ and $k = -0.931201$. Stop the continuation before the GH point, e.g. for $\lambda = 1.6749071$ where $k = -0.32998453$ and $x = y = z = 0.98236575$. From this Hopf point start the continuation of limit cycles (this is the default Curve type) with $ntst = 50$, $ncol = 4$ and amplitude equal to $1e-5$.

We then obtain limit cycles for slowly decreasing values of λ. At $\lambda = 1.6748397$ the parameter λ does not decrease further (at least as seen for this number of digits) but the period increases at each step by approximately the amount of MaxStepsize.

Stop the continuation when the period reaches 95.02325 (or a nearby value).

Now select the last LC of this curve (Select → Initial point), and declare it to be a homoclinic orbit by clicking on Type → Curve → Homoclinic to Saddle.

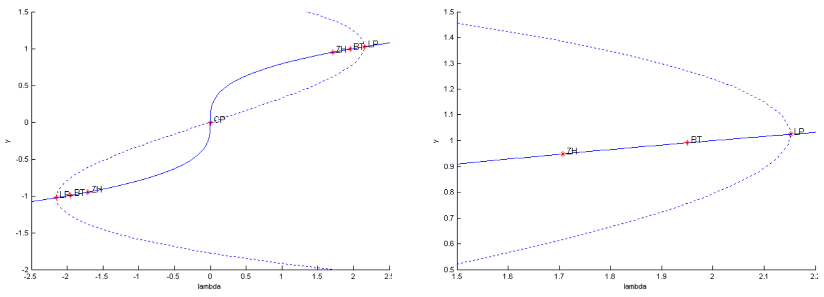


Figure 31: Left: equilibrium bifurcation diagram of the Koper system. Dashed line = equilibria, full line = limit points, CP = Cusp Point, BT = Bogdanov-Takens, ZH = Zero-Hopf, LP = Limit Point. Right: zoom on the right part of the diagram.

For a homoclinic continuation you need 2 free parameters (here k and λ) and 1 or 2 homoclinic parameters. Pick ϵ_0 and ϵ_1. With these settings, you can start the homoclinic curve (Compute $\rightarrow$ Backward). See Figure 32. For speed purposes, you can increase some Tolerances in the Continuer window, or set Adapt to 0.

One can also monitor the eigenvalues of the equilibrium during continuation, by displaying them in the Numerical window. This is a useful feature, because it gives indications on what further bifurcations might be expected. For example, a non-central homoclinic-to-saddle-node reveals itself by the fact that one eigenvalue approaches zero.

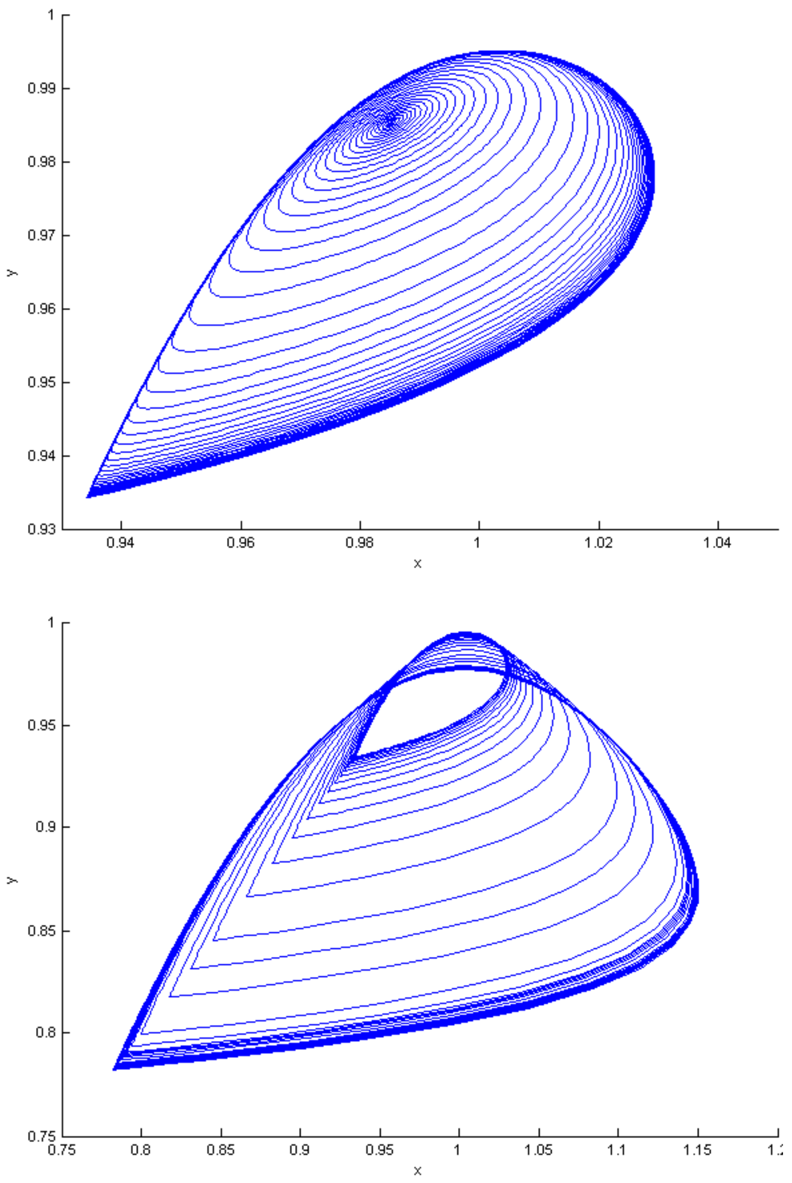


Figure 32: Top: limit cycles, starting from a Hopf point H and approaching a homoclinic orbit. Bottom: a family of HHS orbits.

References

[1] E.L. Allgower and K. Georg, Numerical Continuation Methods: An introduction, Springer-Verlag, 1990 .

[2] Beyn, W.-J. 1994. Numerical analysis of homoclinic orbits emanating from a Takens-Bogdanov point. IMA J. Numerical Analysis, 14, 381-410.

[3] W.J. Beyn, A. Champneys, E. Doedel, W. Govaerts, Yu.A. Kuznetsov, and B. Sandstede, Numerical continuation and computation of normal forms. In: B. Fiedler, G. Iooss, and N. Kopell (eds.) “Handbook of Dynamical Systems : Vol 2”, Elsevier 2002, pp 149 - 219.

[4] Champneys, A.R. and Kuznetsov Yu.A. 1994. Numerical detection and continuation of codimension-two homoclinic orbits. Int. J. Bifurcation Chaos, 4(4), 785-822.

[5] Champneys, A.R., Kuznetsov Yu.A. and Sandstede B. 1996. A numerical toolbox for homoclinic bifurcation analysis. Int. J. Bifurcation Chaos, 6(5), 867-887.

[6] C. De Boor and B. Swartz, Collocation at Gaussian points, SIAM Journal on Numerical Analysis 10 (1973), pp. 582-606.

[7] Demmel, J.W., Dieci, L. and Friedman, M.J. 2001. Computing connecting orbits via an improved algorithm for continuing invariant subspaces. SIAM J. Sci. Comput., 22(1), 81-94.

[8] A. Dhooge, W. Govaerts and Yu. A. Kuznetsov, MATCONT : A matlab package for numerical bifurcation analysis of ODEs, ACM Transactions on Mathematical Software 29(2) (2003), pp. 141-164.

[9] E. Doedel and J Kernévez, AUTO: Software for continuation problems in ordinary differential equations with applications, California Institute of Technology, Applied Mathematics, 1986.

[10] Doedel, E.J. and Friedman, M.J.: Numerical computation of heteroclinic orbits, J. Comp. Appl. Math. 26 (1989) 155-170.

[11] E.J. Doedel, A.R. Champneys, T.F. Fairgrieve, Yu.A. Kuznetsov, B. Sandstede and X.J. Wang, auto97-00 : Continuation and Bifurcation Software for Ordinary Differential Equations (with HomCont), User’s Guide, Concordia University, Montreal, Canada (1997-2000). (`http://indy.cs.concordia.ca`).

[12] Doedel, E.J., Govaerts W., Kuznetsov, Yu.A.: Computation of Periodic Solution Bifurcations in ODEs using Bordered Systems, SIAM Journal on Numerical Analysis 41,2(2003) 401-435.

[13] Doedel, E.J., Govaerts, W., Kuznetsov, Yu.A., Dhooge, A.: Numerical continuation of branch points of equilibria and periodic orbits, (preprint 2003) .

[14] Ermentrout, B.: Simulating, Analyzing, and Animating Dynamical Systems. Siam Publications, Philadelphia, 2002.

[15] Freire, E., Rodriguez-Luis, A., Gamero E. and Ponce, E., A case study for homoclinic chaos in an autonomous electronic circuit: A trip form Takens-Bogdanov to Hopf- Shilnikov, Physica D 62 (1993) 230–253.

[16] Friedman, M., Govaerts, W., Kuznetsov, Yu.A. and Sautois, B. 2005. Continuation of homoclinic orbits in MATLAB. LNCS, 3514, 263-270.

[17] Genesio, R. and Tesi, A. Harmonic balance methods for the analysis of chaotic dynamics in nonlinear systems. Automatica 28 (1992), 531-548.

[18] Genesio, R., Tesi, A., and Villoresi, F. Models of complex dynamics in nonlinear systems. Systems Control Lett. 25 (1995), 185-192.

[19] W.J.F. Govaerts, Numerical Methods for Bifurcations of Dynamical Equilibria, SIAM, 2000.

[20] Govaerts, W. and Sautois, B.: Phase response curves, delays and synchronization in MATLAB. LNCS, 3992 (2006), 391-398.

[21] Govaerts, W. and Sautois, B.: Computation of the phase response curve: a direct numerical approach. Neural Comput. 18(4) (2006), 817-847.

[22] KOPER, M. 1995. Bifurcations of mixed-mode oscillations in a three-variable autonomous Van der Pol-Duffing model with a cross-shaped phase diagram. Phys. D, 80, 72-94.

[23] Yu.A. Kuznetsov, Elements of Applied Bifurcation Theory, Springer-Verlag, 1998.

[24] Yu.A. Kuznetsov and V.V. Levitin, CONTENT: Integrated Environment for analysis of dynamical systems. CWI, Amsterdam 1997: `ftp://ftp.cwi.nl/pub/CONTENT`

[25] MATLAB, The Mathworks Inc., `http://www.mathworks.com`.

[26] W. Mestrom, Continuation of limit cycles in MATLAB, Master Thesis, Mathematical Institute, Utrecht University, The Netherlands, 2002.

[27] Morris, C., Lecar,H., Voltage oscillations in the barnacle giant muscle fiber,Biophys J. 35 (1981) 193–213.

[28] A. Riet, A Continuation Toolbox in MATLAB, Master Thesis, Mathematical Institute, Utrecht University, The Netherlands, 2000.

[29] D. Roose et al., Aspects of continuation software, in : Continuation and Bifurcations: Numerical Techniques and Applications, (eds. D. Roose, B. De Dier and A. Spence), NATO ASI series, Series C, Vol. 313, Kluwer 1990, pp. 261-268.

[30] Terman, D., Chaotic spikes arising from a model of bursting in excitable membranes, Siam J. Appl. Math. 51 (1991) 1418–1450.

[31] Terman, D., The transition from bursting to continuous spiking in excitable membrane models, J. Nonlinear Sci. 2, (1992) 135–182.