CL_MATCONTL: a continuation toolbox for large equilibrium problems in MATLAB.

David Bindel[1],
Willy Govaerts[2],
Jeremy Hughes[3],
Yuri A. Kuznestov[4],
Mark Pekkér (formerly Mark Friedman)[3],
Daniel Veldman[4]

[1] *Department of Computer Science, Cornell University, Ithaca, NY 14853, United States.*
[2] *Department of Applied Mathematics and Computer Science, Ghent University, Krijgslaan 281-S9, B-9000 Ghent, Belgium.*
[3] *Mathematical Sciences Department, University of Alabama in Huntsville, Huntsville, AL 35899, United States. The author was supported in part under NSF DMS-0209536 and NSF ATM-0417774.*
[4] *Mathematisch Instituut, Utrecht University, Budapestlaan 6, 3584CD Utrecht, The Netherlands.*

October, 2015

Abstract

CL_MATCONT and its GUI version MATCONT are MATLAB packages for the study of dynamical systems and their bifurcations for small and moderate size problems. CL_MATCONTL is an extension of the functionality of CL_MATCONT to large-scale computations of equilibria bifurcations, via subspace reduction. It allows to compute curves of equilibria, limit points, Hopf points, and Bogdanov-Takens points and supports branch switching. CL_MATCONTL can also monitor user - defined functions and locate their roots along computed curves. It can compute all necessary derivatives by finite differences or from file.

In this tutorial we consider two examples. The first example is a finite difference discretization of the 1D Brusselator, a well known model system for autocatalytic chemical reactions with diffusion. The second example is a 1D transport model of a fusion plasma. Most of the capabilities of CL_MATCONTL are illustrated.

Contents

1 Introduction

1.1 Overview of Capabilities

Consider a parametrized dynamical system defined as a set of ODEs of the form

$$\frac{du}{dt} = f(u, \alpha), \quad f(u, \alpha), u \in \mathbb{R}^n, \tag{1}$$

where α denotes one or more free parameters. The aim of CL_MATCONTL is to provide a continuation environment in MATLAB for branches of large stationary problems

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

which is compatible with the standard MATLAB ODE representation. A typical example comes from a spatial discretization of elliptic partial differential equations, in which case f_u will typically be large and sparse. In this case, an invariant subspace $\mathcal{R}(s)$ corresponding to a few eigenvalues near the imaginary axis provides information about stability and bifurcations.

CL_MATCONT [10] and its GUI version MATCONT [9] are MATLAB packages for the study of dynamical systems and their bifurcations for small and moderate size problems.

The CIS algorithm is an algorithm for computing a smooth orthonormal basis for an invariant subspace $\mathcal{R}(s)$ of a parameter-dependent matrix $A(s)$ [8, 11, 15, 2], [3], [4]. This allows for computation and continuation of a much smaller restriction $C(s) := A(s)|_{\mathcal{R}(s)}$ of $A(s)$ onto $\mathcal{R}(s)$. To study bifurcations of equilibria we apply the CIS algorithm to $A(x(s)) = f_u(x(s))$ with $\mathcal{R}(s)$ the subspace corresponding to the left most eigenvalues of $A(x(s))$.

We have incorporated the CIS algorithm into CL_MATCONT to extend its functionality to large-scale computations of equilibria bifurcations, via subspace reduction, see [2], [5], [16], [4], [17], [19], for some initial results. This has resulted in a MATLAB bifurcation package CL_MATCONTL [7]. The complete description of the algorithms and examples are given in [6]. This toolbox is developed with the following targets in mind:

- detection of singularities via test functions;
- singularity-specific location code;
- processing of regular and singular points;
- support of derivatives from file;
- support for sparse matrices.

The following actions are supported by the present version of CL_MATCONTL:

- continuation of equilibria with respect to a control parameter;
- detection of fold, Hopf and branching points on curves of equilibria;

- computation of normal form coefficients for fold and Hopf equilibrium bifurcations;
- continuation of fold and Hopf equilibrium bifurcations in two control parameters;
- detection of all codim 2 equilibrium bifurcations (cusp, Bogdanov-Takens, generalized Hopf, zero-Hopf, and double Hopf) on fold and Hopf curves;
- computation of the critical normal form coefficients for codim 2 equilibrium bifurcations, which include multi-linear forms of order up to 5;
- continuation of Bogdanov-Takens bifurcations in three control parameters;
- detection of all codim 3 degeneracies along the Bogdanov-Takens curve (double and triple equilibrium, triple zero and Bogdanov-Takens/Hopf);
- computation of the critical normal form coefficients for the double and triple equilibrium (see [21]);
- branch switching at branch points of equilibria using 3 methods.

The following notation is used for singular points on curves CL_MATCONTL:

H a Hopf point on a curve of equilibria

BP a branch point on a curve of equilibria

LP a limit point on a curve of equilibria

BT a Bogdanov-Takens point on a curve of limit points or Hopf points

ZH a zero-Hopf point on a curve of limit points or Hopf points

CP a cusp point on a curve of limit points

BP a branch point on a curve of limit points

DH a double-Hopf point on a curve of Hopf points

GH a generalized Hopf, or Bautin point on a curve of Hopf points

BTa2 a triple equilibrium degeneracy on a curve of Bogdanov-Takens bifurcations

BTb2 a double equilibrium degeneracy on a curve of Bogdanov-Takens bifurcations

BT3 a triple zero degeneracy on a curve of Bogdanov-Takens bifurcations

BTH a Bogdanov-Takens/Hopf degeneracy on a curve of Bogdanov-Takens bifurcations

00 the first point on the curve

99 the last point on the curve

We assume that the reader is familiar with dynamical systems theory and knows about equilibria and codimension-1 and 2 bifurcations, see [18], [20]. We also assume an elementary knowledge of MATLAB but do not assume familiarity with numerical methods.

The continuation of the curve of Bogdanov-Takens bifurcations allows to detect codimension-3 degeneracies. These degeneracies provide additional insight in the dynamics on the center manifold of the system (1), see [1], [13], [21], [14], [12]. This can provide more insight in the arrangement of codimension-1 and 2 bifurcations nearby.

1.2 Installation and Initialization

The current version of the package is freely available for download at:

`http://uah.edu/faculty/pekker`

CL_MATCONTL requires MATLAB 7.* ?? to be installed on your computer. When using a 64-bit version of MATLAB, you should also make sure that you have a C-compiler installed.

Downloading and unzipping the CL_MATCONTL package (for example in the MATLAB 'work' directory) creates a directory 'CL_MATCONTL' on the computer. It contains `init.m` and `ReadMe.txt` files, as well as a number of subdirectories.

We start CL_MATCONTL by changing to the 'CL_MATCONTL' directory and typing '`init`' on the MATLAB command line. The `init.m` file registers the directories with MATLAB and compiles any .mex files if necessary. If mex files have never been compiled on your machine, MATLAB may prompt you to select a compiler for `.mex` files. You must run `init.m` each time you open MATLAB when you wish to use CL_MATCONTL.

1.3 User supplied Files.

The user must prepare the following two m-files.

The problem definition m-file (see Section 3.1) contains functions that define a particular problem, including the function f in equation (2), the symbolic Jacobian matrix, etc.

The run script (see Section 3.2) contains the initial parameters for the problem, the options to be used, and the code that executes continuation.

We advise to create for every problem a new folder in the `/User/` directory.

2 Tutorials

2.1 The 1D Brusselator

The *1D Brusselator* is a well known model system, intended to model the Belusov - Zhabotinsky reaction, for autocatalytic chemical reactions with diffusion:

$$\begin{array}{ll} \frac{d_1}{l^2}u'' - (b+1)u + u^2v + a = \dot{u}, & \frac{d_2}{l^2}v'' + bu - u^2v = \dot{v}, \quad \text{in } \Omega = (0,1), \\ u(0) = u(1) = a, & v(0) = v(1) = \frac{b}{a}. \end{array} \tag{3}$$

This problem exhibits a rich bifurcation scenario and has been used in the literature as a standard model for bifurcation analysis. We utilize the second order central difference discretization

$$f'' \approx \frac{1}{h^2}(f_{i-1} - 2f_i + f_{i+1}), \quad h = (N+1)^{-1},$$

with uniform grid of N grid points. Since there are two unknowns per grid point, the resulting discrete problem, which has dimension $n = 2N$, can be written in the form (2). This discretization of the Brusselator is used in a CL_MATCONT example [10]. To avoid spurious solutions (solutions that are induced by the discretization but do not actually correspond to solutions of the continuous problem), one can vary 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 Jacobian matrix is a sparse 5-band matrix. In the problem definition file `Tutorial/bruss_1d/bruss_1d.m`, the Jacobian matrix is introduced as a sparse matrix.

Except for Section 2.1.5, we we use an approximate nonconstant initial equilibrium solution:

$$\left\{ \begin{array}{lcl} u(x) & = & a + 2\sin(\pi x), \\ v(x) & = & \frac{b}{a} - \frac{1}{2}\sin(\pi x). \end{array} \right. \tag{4}$$

This solution (4) is not an equilibrium, but CL_MATCONTL will find an equilibrium close to this initial solution. The initial values of the parameters are: $a = 1$, $b = 17.1$, $d_1 = 0.0016$, $d_2 = 0.08$, and $l = 0.06$. In this demo we use $N = 500$, hence the size of the system (2) is $n = 1000$.

The files for this problem are located in `/Tutorial/bruss_1d`. You should make this folder your 'current directory' in MATLAB for this tutorial. Also, make sure you have runned `init.m` before starting the tutorial (see Section 1.2).

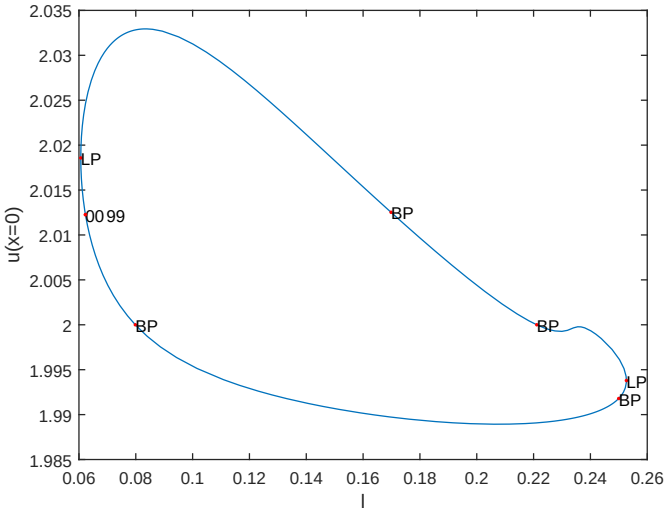

Figure 1: `testbruss_BP0`, continuation of an equilibrium branch.

2.1.1 `testbruss_BP0`: Continuation of an equilibrium

We perform continuation of the equilibrium solution curve with the parameter l free, by typing `testbruss_BP0` in MATLAB command line. This produces the following output in the MATLAB command window:

```
>> testbruss_BP0
 S ID  PT:  p(2)            ||u||          ||f||
  1  1  00:  +6.228439e-02  8.675355e+01  7.934112e-10
 12  2  BP:  +7.984536e-02  6.815405e+01  2.548914e-08
 62  3  BP:  +2.499571e-01  7.523826e+01  3.947069e-10
 67  4  LP:  +2.526672e-01  7.453336e+01  7.217252e-10
 85  5  BP:  +2.211210e-01  6.815424e+01  6.556548e-08
 95  6  BP:  +1.698320e-01  7.775931e+01  2.000414e-08
161  7  LP:  +6.063973e-02  9.901921e+01  2.870064e-09
167  8  99:  +6.228439e-02  8.675355e+01  7.934112e-10

Elapsed time  = 9.1 secs
Number of Points = 167
```

The computed curve is also visualized by the graph in Figure 1, which is also created when running `testbruss_BP0`. From these outputs we see that four branching points (BP) and two Limit Points (LP) are located along the curve. Also note that the continuation is stopped when the starting point is reached again at step 167.

Furthermore, `/Data/` and `/Logs/` directories have been created inside the `/testbruss_1d/` directory containing the data of the computed curves. See Section 3.2.4 for more information.

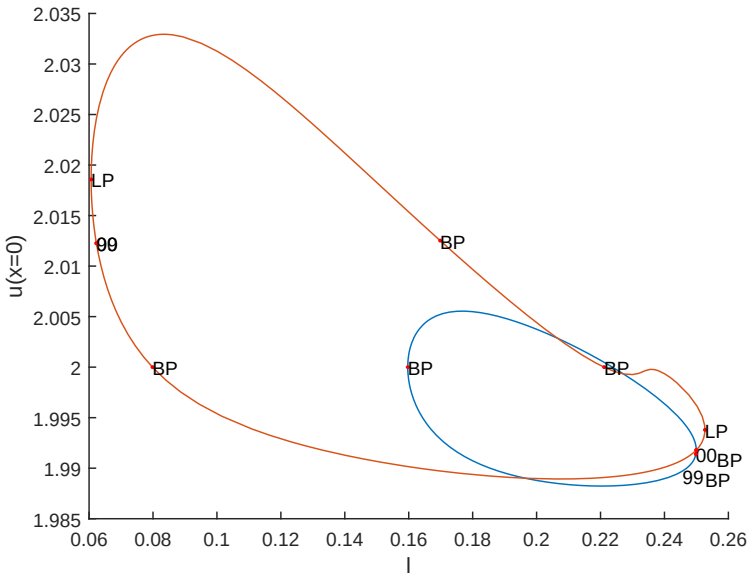

Figure 2: `testbruss_BP0` and `testbruss_BP1`, branch switching.

2.1.2 `testbruss_BP1`: Branch switching

When a branch point is located, CL_MATCONTL attempts to compute a tangent vector to the second branch. If successful, this tangent vector is saved in the `.mat` file. By using `init_BP_EP_L`, this tangent vector is used to continue the second branch.

We restart at BP computed at Step 62 with `ID`=3 of the `testbruss_BP0` run from Section 2.1.1. We switch to the second branch, and perform its continuation by running the test file `testbruss_BP1`. This produces the output:

```
>> testbruss_BP1
 S ID  PT:  p(2)            ||u||          ||f||
  1  1  00:  +2.498387e-01  7.522087e+01
 50  2  BP:  +1.596876e-01  6.815424e+01  2.532307e-07
101  3  BP:  +2.499571e-01  7.523826e+01  2.822453e-08
104  4  99:  +2.498387e-01  7.522087e+01

Elapsed time  = 6.1 secs
Number of Points = 104
```

Note that the run stops at step 104 because the starting point is reached again. Also Figure 2 is produced by the combination of the `testbruss_BP0` and `testbruss_BP1` runs.

Remark 1. The tangent vector to the second branch can be obtained in one of three ways:

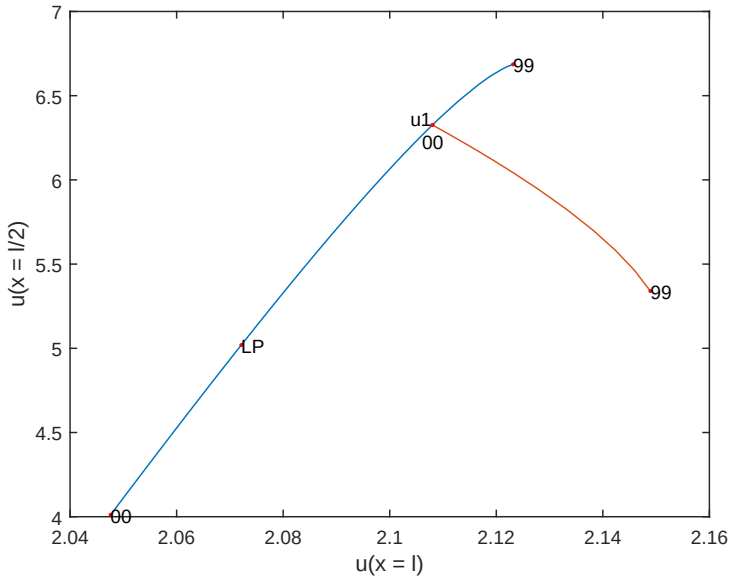

Figure 3: `testbruss_U0` and `testbruss_U1`, restarting from a user point.

1. Calculating the normal form coefficients and then finding a tangent vector to the second branch by using the algebraic branching equation (ABE).
2. Finding an orthogonal vector to the tangent vector to the first branch, in the Null space of the Jacobian matrix at the branch point, and using it as an initial guess for a tangent vector to the second branch.
3. A bisection-like procedure, which is an improved version of 2.

`testbruss_BP0` uses the third method.

2.1.3 `testbruss_U1`: Restarting at a user point

A user point is defined by a user function in a problem definition m-file. Once a user point is located and saved, it can be used as the starting point for continuation of another curve. This is illustrated by the files `testbruss_U0` and `testbruss_U1`.

We perform continuation of the equilibrium solution curve (u, l), where l is a free parameter, by typing `testbruss_U0` in the MATLAB command window. This produces the following output:

```
>> testbruss_U0
 S ID  PT:  p(2)            ||u||          ||f||
  1  1  00:  +6.228294e-02  4.394268e+01  7.847343e-10
 10  2  LP:  +6.063833e-02  5.017628e+01  3.151398e-10
 22  3  u1:  +6.499999e-02  5.932165e+01  1.869982e-07
 30  4  99:  +7.279746e-02  6.229677e+01  1.811860e-08

Elapsed time  = 1.0 secs
Number of Points = 30
```

Here the located root of the user function is labeled `u1`. Running `testbruss_U0` produces the blue curve in Figure 3.

We start continuation from the user point in the parameter a by typing `testbruss_U1` in the MATLAB command window. This produces the following output:

```
>> testbruss_U1
 S ID  PT:  p(3)            ||u||          ||f||
  1  1  00:  +2.000000e+00  5.932165e+01  1.869982e-07
 15  2  99:  +2.069486e+00  5.223176e+01  1.542012e-07

Elapsed time  = 0.4 secs
Number of Points = 15
```

The orange curve in Figure 3 is computed.

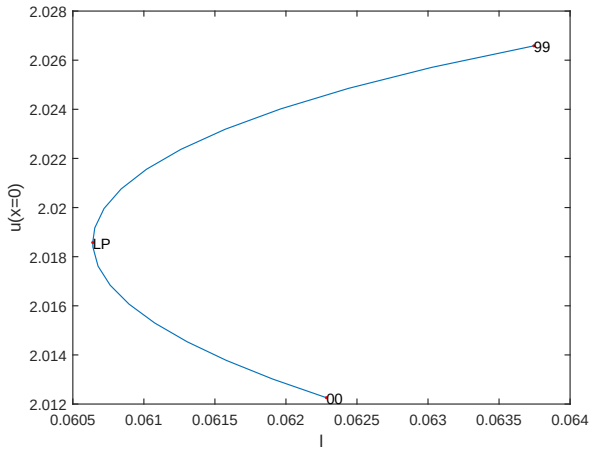

Figure 4: `testbruss_LP0`, continuation of an equilibrium branch.

2.1.4 `testbruss_LP1`: Continuation of limit points

Along the equilibrium curve computed from `testbruss_BP0` in Section 2.1.1, a Limit Point (LP) is located at Step 161 for $l = 0.060640$. We reduce the number of steps needed to find this LP by changing by changing the continuation direction. This results in `testbruss_LP0.m`. Running `testbruss_LP0.m` produces the following output,

```
>> testbruss_LP0
 S ID  PT:  p(2)            ||u||          ||f||
  1  1  00:  +6.228439e-02  8.675355e+01  7.934112e-10
 10  2  LP:  +6.063973e-02  9.901921e+01  2.752822e-09
 20  3  99:  +6.374478e-02  1.147716e+02  3.237868e-07

Elapsed time  = 1.1 secs
Number of Points = 20
```

See Figure 4 for the bifurcation diagram.

We restart at the LP computed at Step 13 with `ID`=2 of the `testbruss_LP0` run, and compute a curve of Limit Points in two free parameters (l, a). The continuation is performed by running `testbruss_LP1`. This produces the output:

```
>> testbruss_LP1
 S ID  PT:  p(2)            p(3)            ||u||          ||f||
  1  1  00:  +6.063973e-02  +2.000000e+00  9.901921e+01  2.075936e-10
 14  2  CP:  +1.102634e-01  +2.549658e+00  6.982917e+01  1.259553e-07
105  3  CP:  +3.671906e-01  +1.581701e+00  9.729630e+01  5.947861e-08
143  4  BT:  +5.466570e-01  +1.045001e+00  1.499048e+02  6.896744e-12
150  5  99:  +6.172540e-01  +9.195194e-01  1.698827e+02  1.703222e-07

Elapsed time  = 30.0 secs
Number of Points = 150
```

The resulting bifurcation diagram is shown in Figure 5. Along the computed curve two Cusp (CP) bifurcations and one Bogdanov-Takens (BT) bifurcation are located. The second CP bifurcation located at step 105 shows the characteristic 'wedge' shape in the two parameter plot, which can be seen in the detailed view in Figure 6.

Remark 2. This 'wedge' cannot be observed at the first CP bifurcation located at step 14. Inspection of the cubic coefficient in the Log-file `testbruss_LP1.txt` shows that $c = -5.723835 \cdot 10^{-4}$. This indicates that the first CP bifurcation is (nearly) degenerate, which explains why the 'wedge' is not observed at this CP bifurcation.

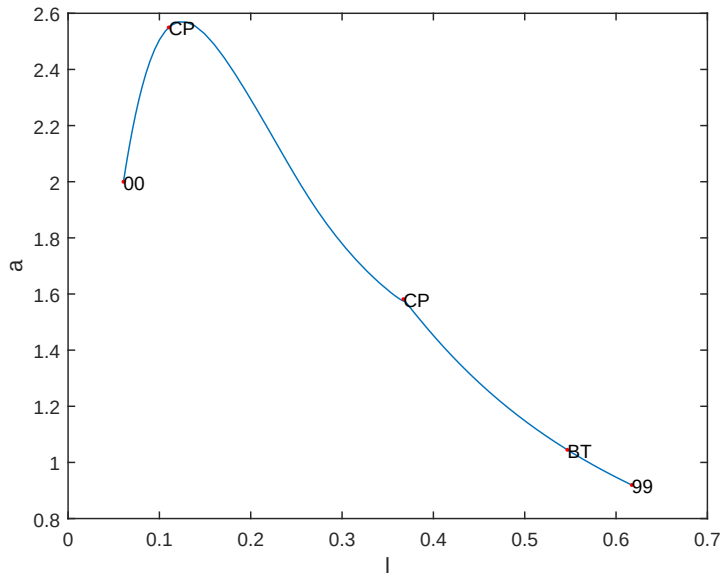

Figure 5: `testbruss_LP1`, continuation Limit Points.

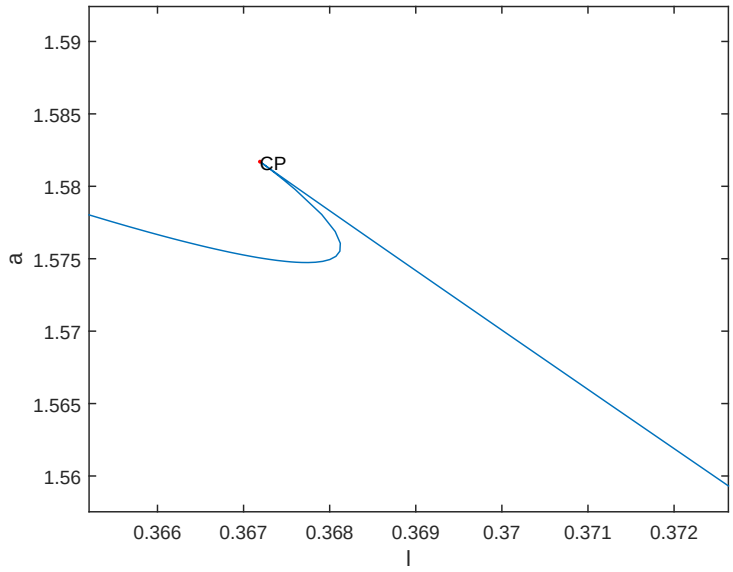

Figure 6: `testbruss_LP1`, detailed view near the second CP point.

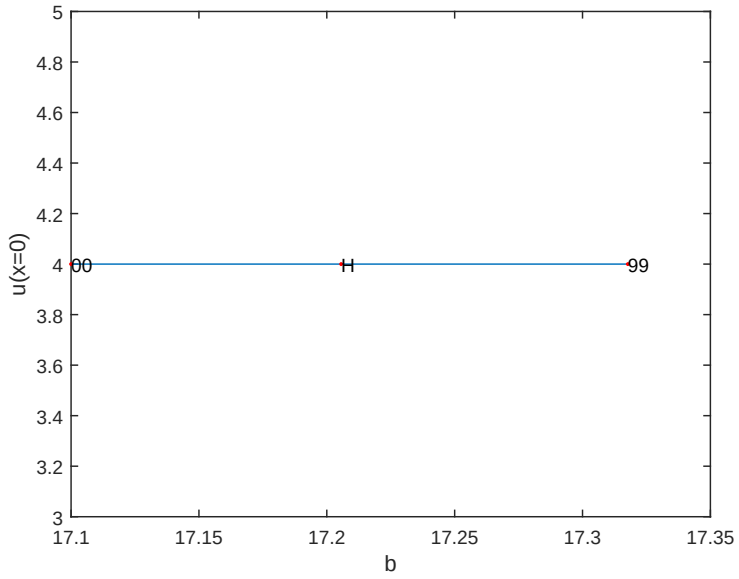

Figure 7: `testbruss_HP0`, continuation of equilibria.

2.1.5 `testbruss_HP1`: Continuation of Hopf points

In this section, we start from a constant equilibrium solution:

$$\begin{cases} u(x) &= a, \\ v(x) &= \frac{b}{a}. \end{cases} \tag{5}$$

The initial values of the parameters are: $a = 4$, $b = 17.1$, $d_1 = 1$, $d_2 = 2$ and, $l = 12$.

We perform continuation of the equilibrium solution curve (u, b), where b is a free parameter, using `testbruss_HP0`. This produces the following output:

```
>> testbruss_HP0
 S ID  PT:  p(4)            ||u||          ||f||
  1  1  00:  +1.710000e+01  8.279568e+01  0.000000e+00
  5  2  H :  +1.720561e+01  8.306873e+01  6.022000e-12
 10  3  99:  +1.731773e+01  8.335945e+01  9.000568e-11

Elapsed time  = 0.7 secs
Number of Points = 10
```

A Hopf point `H` is located at Step 5 with `ID`=2. Also Figure 7 is produced.

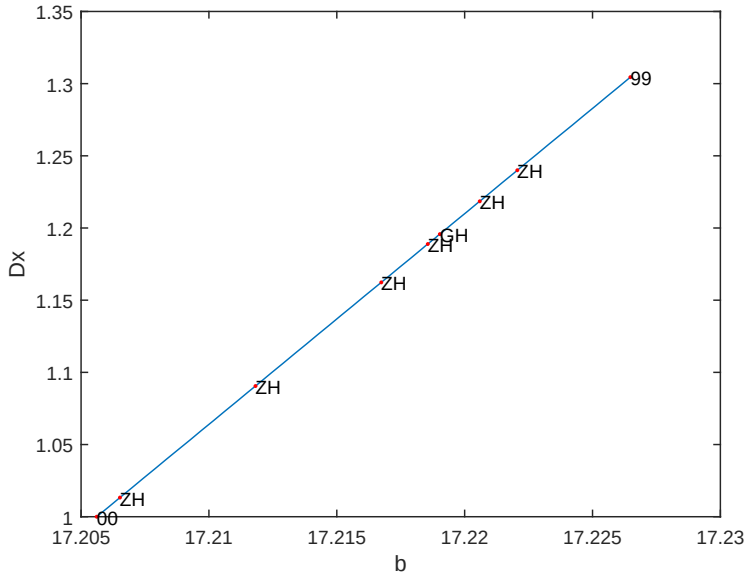

Figure 8: `testbruss_HP1`, continuation of Hopf Points.

We then restart at `H` and compute a curve of Hopf points in two control parameters (b, d_1) using `testbruss_HP1`. This produces the output:

```
>> testbruss_HP1
 S ID  PT:  p(4)            p(5)            ||u||          ||f||
  1  1  00:  +1.720561e+01  +1.000000e+00  8.306873e+01  1.445879e-07
  3  2  ZH:  +1.720652e+01  +1.013273e+00  8.307108e+01  1.445869e-07
 16  3  ZH:  +1.721182e+01  +1.090524e+00  8.308479e+01  1.445912e-07
 28  4  ZH:  +1.721674e+01  +1.162332e+00  8.309754e+01  1.445913e-07
 37  5  ZH:  +1.721856e+01  +1.188899e+00  8.310225e+01  1.445921e-07
 49  6  GH:  +1.721903e+01  +1.195760e+00  8.310347e+01  1.445918e-07
 56  7  ZH:  +1.722059e+01  +1.218499e+00  8.310751e+01  1.445935e-07
 60  8  ZH:  +1.722206e+01  +1.240007e+00  8.311133e+01  1.445590e-07
 70  9  99:  +1.722648e+01  +1.304493e+00  8.312278e+01  1.445570e-07

Elapsed time  = 10.4 secs
Number of Points = 70
```

Along the curve six Zero-Hopf (ZH) bifurcations and one Generalized Hopf (GH) or Bautin bifurcation are located. The curve is shown in Figure 8.

Remark 3. Inspection of Log-file shows that there are six unstable eigenvalues at the starting point. At each ZH bifurcation one of these eigenvalues moves through the imaginary axis. At the endpoint there are no unstable eigenvalues.

2.2 A 1D transport model of a fusion plasma

In this section, we consider a finite difference discretization of a one dimensional transport model of a fusion plasma in TOKAMAK reactor. The model was orignally proposed in [25]. The transport model consists of three coupled PDEs describing the particle density n, the plasma temperature T and the radial electric field or poloidal rotation Z:

$$\frac{\partial n}{\partial t} = -\frac{\partial \Gamma}{\partial x} \qquad \Gamma = -D(Z)\frac{\partial n}{\partial x}, \tag{6}$$

$$\frac{\partial U}{\partial t} = -\frac{\partial q}{\partial x} \qquad q = -\chi(Z)n\frac{\partial T}{\partial x} + \frac{\Gamma T}{\gamma - 1}, \tag{7}$$

$$\varepsilon\frac{\partial Z}{\partial t} = \mu\frac{\partial^2 Z}{\partial x^2} + c_n\frac{T}{n^2}\frac{\partial n}{\partial x} + \frac{c_T}{n}\frac{\partial T}{\partial x} + G(Z), \tag{8}$$

where

$$D(Z) = D_0 + D_1\tanh(Z), \tag{9}$$

$$\chi(Z) = \frac{1}{(\gamma-1)\zeta}D(Z), \tag{10}$$

$$G(Z) = a - b(Z - Z_S) - (Z - Z_S)^3. \tag{11}$$

The internal energy U is related to temperature T and density n by

$$U := \frac{nT}{\gamma - 1}. \tag{12}$$

The following boundary conditions are used for the problem

$$\Gamma|_{x=\infty} = \Gamma_\infty, \qquad q|_{x=\infty} = q_\infty, \qquad \left.\frac{\partial Z}{\partial x}\right|_{x=\infty} = 0, \tag{13}$$

$$\left.\frac{\partial n}{\partial x}\right|_{x=0} = \frac{n}{\lambda_n}, \qquad \left.\frac{\partial T}{\partial x}\right|_{x=0} = \frac{T}{\lambda_T}, \qquad \left.\frac{\partial^2 Z}{\partial x^2}\right|_{x=0} = 0. \tag{14}$$

We use a 3-point finite difference discretization on the nonuniform grid

$$\Theta := \{0 = x_0 < x_1 < x_2, \ldots, < x_N = L\}, \qquad x_i = L\,(i/N)^3\,. \tag{15}$$

The finite difference formulas can be found in (for example) [18]. This leads to a system of $n = 3(N-1)$ ODEs, which are specified in the `fun_eval` function in the file `Tutorial/fusion/fusion.m`.

The parameter values at the starting point are $\Gamma_\infty = -0.8$, $q_\infty = -0.008$, $D_0 = 1.9$, $D_1 = -1.1$, $a = -0.01$, $b = -0.01$, $\zeta = 1.1$, $\mu = 0.05$, $\varepsilon = 0.05$, $Z_S = 0$, $\gamma = 5/3$, $\lambda_n = 1.25$, $\lambda_T = 1.5$, $c_n = 1.1$, and $c_T = 0.9$. In this tutorial, we use $N = 50$ so the size of the system is $n = 147$.

2.2.1 `testfusion_BT2`: Continuation of Bogdanov-Takens points

The dynamics of this model have been studied in [23], [22], and [24]. These studies suggested the presence of a degenerate Bogdanov-Takens in the model. In this tutorial we will locate a degenerate Bogdanov-Takens bifurcation in this model by continuation of Bogdanov-Takens bifurcations.

The files for this problem are located in the `Tutorial/fusion` directory. Make this your 'current directory' in MATLAB. Also make sure that `init.m` has been runned before starting the tutorial (see Section 1.2).

To find a starting point on the Bogdanov-Takens curve, we first locate a Limit Point and continue this Limit Point to find a Bogdanov-Takens bifurcation. Alternatively, a starting poin could be found by continuation of a Hopf point.

We find the Limit Point by running `testfusionBT0`. This produces the following output

```
>> testfusion_BT0
 S ID  PT:  p(3)            ||u||          ||f||
  1  1  00:  -7.989681e-03  1.291934e+01  2.747427e-08
 12  2  H :  -7.415397e-03  1.273614e+01  9.525691e-09
 14  3  LP:  -7.413592e-03  1.272351e+01  7.034363e-10
 20  4  99:  -7.472975e-03  1.263951e+01  5.055879e-09

Elapsed time  = 76.7 secs
Number of Points = 20
```

Note that the located Hopf and Limit Point are close, which is natural near a Bogdanov-Takens bifurcation. Also the graph in Figure 9 is created.

Remark 4. Observe that the handle `jacobian` in the problem definition file `Tutorial/fusion/fusion.m` is empty for this problem, so that the jacobian is computed using finite differences. For this problem, this part is most time consuming. In this example we used the MATLAB's Parallel Computing Toolbox to speed up the computation. The use of the Parallel Computing Toolbox can be enables using the option `cont_ParallelComputing`, see Section 3.2.5.

To find the Bogdanov-Takens we continue the located Limit Point by running `testfusionBT1`. This produces the following output:

```
>> testfusion_BT1
 S ID  PT:  p(3)            p(8)            ||u||          ||f||
  1  1  00:  -7.413592e-03  -1.000000e-02  1.272351e+01  3.735505e-09
  8  2  BT:  -4.549683e-03  -5.056140e-02  1.273870e+01  4.061119e-09
 10  3  99:  -2.993005e-03  -6.474749e-02  1.271994e+01  2.161779e-08

Elapsed time  = 155.2 secs
Number of Points = 10
```

Also the graph in Figure 10 is created.

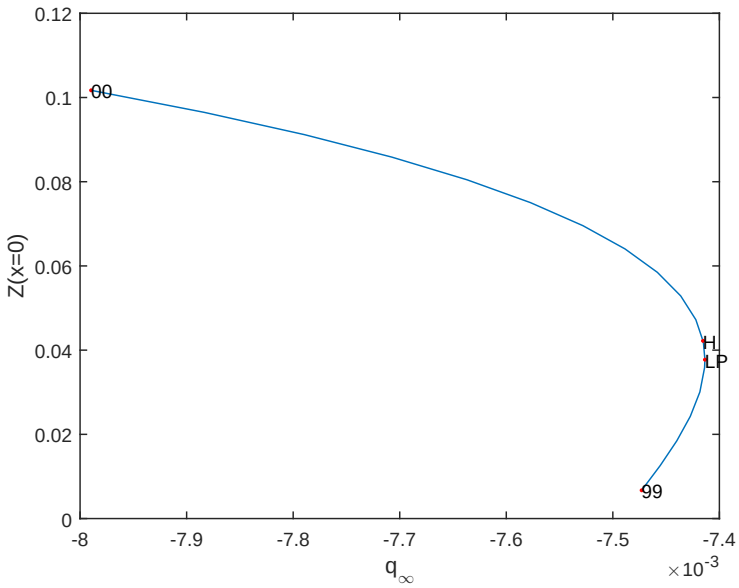

Figure 9: `testfusion_BT0`, continuation of equilibria.

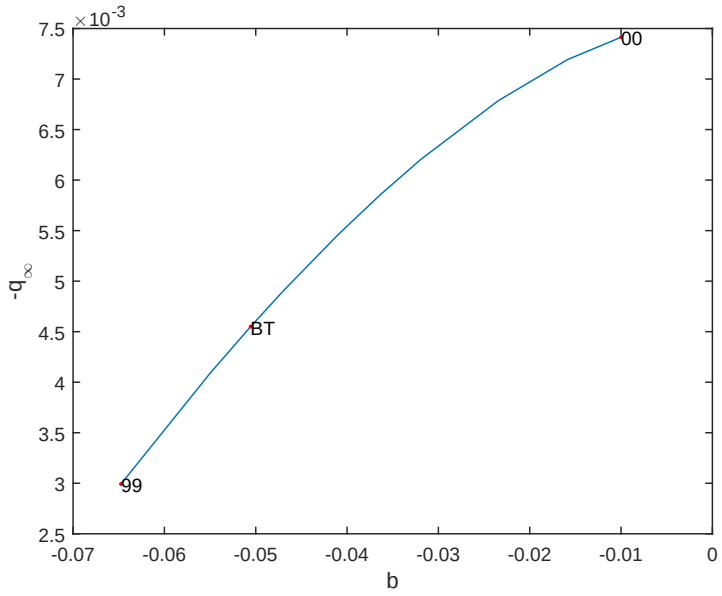

Figure 10: `testfusion_BT1`, continuation of Limit Points.

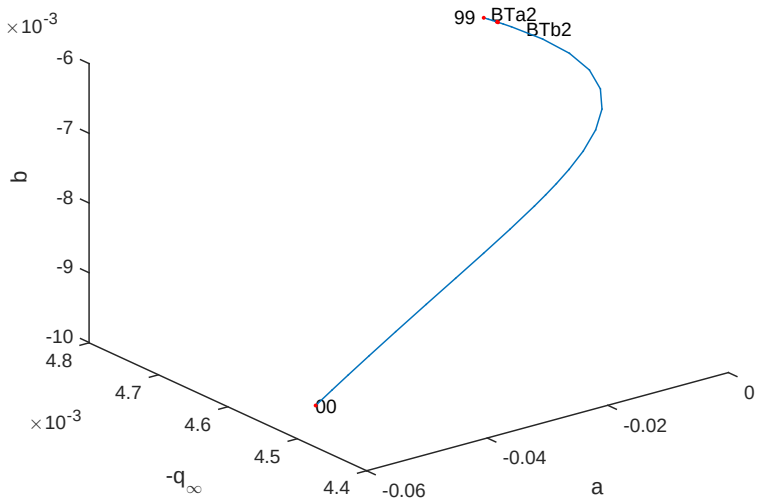

Figure 11: `testfusion_BT2`, continuation of Bogdanov-Takens Points.

Now the located Bogdanov-Takens bifurcation is continued in three parameters by running `testfusionBT2`. This produces the following output:

```
>> testfusion_BT2
S ID  PT:  p(3)           p(7)           p(8)           ||u||         ||f||
 1  1  00:  -4.555658e-03 -1.000513e-02 -5.054334e-02 1.273861e+01 6.049360e-08
24  2  BTb2:-4.700360e-03 -6.272584e-03 -3.638102e-03 1.278346e+01 2.107071e-08
26  3  BTa2:-4.702086e-03 -6.274886e-03 -3.639142e-03 1.278247e+01 9.114035e-08
30  4  99:  -4.720506e-03 -6.299384e-03 -3.688477e-03 1.277170e+01 1.071080e-08

Elapsed time  = 633.7 secs
Number of Points = 30
```

Also the graph in Figure 11 is created. Observe that the BTa2 and BTb2 bifurcation are very close to each other.

Remark 5. The fact that the BTa2 and BTb2 are very close to eachother seems to indicate that we are in fact close to the (codimension-4) situation where both normal form coefficients a_2 and b_2 vanish simultaneously. degeneracy in the three parameter continuation. The appearance of this codimension-4 situation is nongeneric in a 3-parameter continuation and the reason for its appearance is not completely understood.

3 User supplied files

3.1 Problem definition m-file

The problem definition m-file describes the particular problem (1). The main function of the problem definition file returns a list of function handles that point to the remaining functions in the file. The main function should have the following structure:

```
function out = PROBDEF
out{1} = @init;
out{2} = @fun_eval
out{3} = @jacobian
out{4} = @jacobianp
out{5} = @usernorm
out{6} = @user1
out{7} = @user2
         ...
```

The handle `@fun_eval` is mandatory. All other handles are optional and can be left empty. As an example, consider `Tutorial/bruss_1d/bruss_1d.m`.

The individual functions are described below. Each function has a specific input and output syntax that must be followed for CL_MATCONTL to be able to use the functions.

3.1.1 init (Optional)

`y0 = init(`$\mathtt{p}_1, ..., \mathtt{p}_n$ `)`

`init` is the first function handle defined in the primary function. It takes the full list of parameters that are being used for the problem and returns the vector `y0`, which must be (a good approximation of) an equilbrium solution at the given parameter values.

3.1.2 fun_eval

`dxdt = fun_eval(t, x,`$\mathtt{p}_1, ..., \mathtt{p}_n$`)`

The function `fun_eval` takes a scalar `t` (which should not be used in the problem), the state vector `x`, and the full parameter list used for the problem. It returns the vector `dxdt`. This notation is used to reflect the nature of the non-linear problem $\dot{x} = f(t, x, p_1, ..., p_n)$.

The function `fun_eval` must be the second function handle returned by the primary function and cannot be left empty.

3.1.3 jacobian (Optional)

`dfdx = jacobian(t, x,`$\mathtt{p}_1, ..., \mathtt{p}_n$`)`

The `jacobian` function takes the same inputs as `fun_eval`. It returns the Jacobian matrix $f_u(t, x, p_1, ..., p_n)$.

If the `jacobian` function is not defined or not being used, the third function handle returned by the primary function must be empty. When this is the case, $f_u(t, x, p_1, ..., p_n)$ is computed using finite differences.

3.1.4 jacobianp (Optional)

`dfdp = jacobianp(t, x, p_1, ..., p_n)`

The `jacobian` function takes the same inputs as `fun_eval`. It returns the Jacobian matrix $f_p(t, x, p_1, ..., p_n)$.

If the `jacobianp` function is not used, the fourth function handle returned by the primary function must be empty. When this is the case, $f_p(t, x, p_1, ..., p_n)$ is computed using finite differences.

3.1.5 Usernorm (Optional)

```
normuser = usernorm(arg)
```

The `usernorm` function takes a vector `arg` and returns a real number. The function has all the properties of a norm. `usernorm` can be used to approximate function norms for ODE and PDE problems.

If the `usernorm` function is not defined or not being used, the fifth function handle returned by the primary function must be empty. In this case the standard Euclidean norm is used.

3.1.6 User Functions (Optional)

`uf1 = user1(t, x, p_1, ..., p_n)`

All remaining function handles are assumed to be User Functions. User functions should take the same inputs as `fun_eval` and return a real number. Roots of (active) user functions will be located by CL_MATCONTL.

For every specified userfunction, there must be an element in the cell array `UserInfo`, containing the following fields

```
UserInfo{i}.name
UserInfo{i}.state
UserInfo{i}.label
```

See Section 3.2.5 for a more detailed discussion of the meaning of the different fields. The structure `UserInfo` must be set by

```
opt = contset(opt,'Cont_UserFuncInfo',UserInfo);
```

This allows CL_MATCONTL to identify and label the user points. An example can be found in `Tutorial/bruss_1d/testbruss_U0`.

3.2 Run script

Test run script files contain all of the information required to compute a curve in CL_MATCONTL. A run script typically consists of the following parts:

Options An option structure is created that containing the options that are used during the continuation of the curve using `contset`.

Initialization A starting point and possible tangent vector are found by a call to `init_**_**_L`.

Continuation The curve is computed by a call to `contL`.

Processing results After continuation, the data of the computed curve accessed using `loadPoint.m`.

This structure resembles the structure used in CL_MATCONT [10]. Below the different parts will be elaborated.

3.2.1 Options

Before specifying options for the specific curve, a new options structure `options` must be created by a call to

```
options = contset();
```

After that, options can then be set using

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

where `optionname` must be a name of an option supported by CL_MATCONTL, which is set to the value specified by `optionvalue`. A list of all options supported by CL_MATCONTL and their default values can be found in Section 3.2.5.

3.2.2 Initialization

Depending on the curve that will be computed and the available initial point, different initialization functions must be used. Currently, the following initialization functions are available.

```
[x0, v0] = init_EP_EP_L(PROBDEF, u, p, ap)
```

`init_EP_EP_L` initializes the continuation of a curve of equilibria, starting from the initial equilibrium `u` at the parameters `p`. `PROBDEF` is a handle to the problem definition file, described in Section 3.1. `ap` is the index of the parameter that is free during the continuation.

When `u` is empty, the initial point at the parameters `p` is found by a call to the function `init` in the problem definition file (see Section 3.1). All other

arguments must be present and cannot be empty. An example of the use of `init_EP_EP_L` can be found in `Tutorial/bruss_1d/testbruss_BP0`.

`init_EP_EP_L` can also be used to start continuation from a previously located user point. See `Tutorial/bruss_1d1/testbruss_U1` for an example.

```
[x0, v0] = init_BP_EP_L(PROBDEF, u, p, ap, data)
```

`init_BP_EP_L` is used for branch switching at a previously located branching point. Currently, branch switching is only supported when the second tangent vector was found during location of the branching point. `data` is the data structure that can be found in the structure `s` from the previously computed equilibrium curve. The active parameter `ap` must be the same as on the previously computed curve.

`PROBDEF` and `data` must always be present. It is recommended to have `u`, `p` and `ap` empty. In this case they are obtained from the data structure and the same as on the previously computed curve. An example of the use of `init_BP_EP_L` can be found in `Tutorial/bruss_1d/testbruss_BP1`.

```
[x0, v0] = init_LP_LP_L(PROBDEF, u, p, ap, data)
```

`init_LP_LP_L` initializes the continuation of a curve of Limit Points, either from a Limit Point located on a previously computed equilibrium curve or, from a point on a previously computed Limit Point curve or, from a previously located Bogdanov-Takens point. `ap` must be a vector of length 2.

When `data` is present, `u`, `p` and `ap` can be left empty. In this case they are obtained from the data structure and the same as on the previously computed curve. When `data` is absent, `u`, `p` and `ap` must be present. `PROBDEF` must always be present. An example of the use of `init_LP_LP_L` can be found in `Tutorial/bruss_1d/testbruss_LP1`.

```
[x0, v0] = init_H_H_L(PROBDEF, u, p, ap, data)
```

`init_H_H_L` initializes the continuation of a curve of Hopf Points, either from a Hopf located on a previously computed equilibrium curve or, from a point on a previously computed Hopf curve. `ap` must be vector of length 2.

`PROBDEF` and `data` must always be present. `u`, `p` and `ap` can be empty. In this case they are obtained from the data structure and the same as on the previously computed curve. An example of the use of `init_H_H_L` can be found in `Tutorial/bruss_1d/testbruss_HP1`.

```
[x0, v0] = init_BT_BT_L(PROBDEF, u, p, ap, data)
```

`init_BT_BT_L` initializes the continuation of a curve of Bogdanov-Takens Points, either from a Bogdanov-Takens point located on a previously computed Limit Point or Hopf curve or a point on a previously computed Bogdanov-Takens curve. `ap` must be vector of length 3.

When `data` is present, `u`, `p` and `ap` can be left empty. In this case they are obtained from the data structure and are the same as on the previously

computed curve. When `data` is not present, `u`, `p` and `ap` cannot be empty. `PROBDEF` must always be present. An example of the use of `init_BT_BT_L` can be found in `Tutorial/fusion/testfusion_BT2`.

3.2.3 contL

```
[s, filename] = contL(CURVEDEF, x0, v0, options);
```

`contL` continues the curve specified by `CURVEDEF`. Currently, the following four curves are available in CL_MATCONTL:

`@equilibriumL`, curve of equilibria.

`@limitpointL`, curve of Limit Point bifurcations.

`@hopfL`, curve of Hopf bifurcations.

`@bogdanovtakensL`, curve of Bogdanov-Takens bifurcations.

`x0` and `v0` should be obtained from the properly chosen initialization function, see Section 3.2.2.

`options` is the options structure discussed in Section 3.2.1.

The function `contL` returns the signgularity sturcture `s` and name of the data file `filename` where the data for that run is stored.

Calling `contL` also creates a `/Data/` folder in the directory `TestPath` This folder contains the data from the continuation in a `.dat` and `.mat` file, named `Filename`. Also a `/Logs/` folder is created in the `TestPath` directory. This folder contains the log-file. `Testpath` and `Filename` can be set using the options structure, see Section 3.2.5. The `.dat` and `.mat` file will be discussed in the next section.

3.2.4 Processing results

The data from the computed curve can be loaded from the `.dat` file in `/Data/` folder using

```
[x, v, h, t, u] = loadPoint(filename)
```

Here `filename` is the name of the `.dat` from which the data is obtained.

The columns of `x` contain the state vector and the active parameters at the different points along the curve. The columns of `v` contain the tangent vectors at the different points along the curve. The vector `h` provides information about the stepsize. When singularities are located, the columns of `t` contain the values of the test functions. When userfunctions are active, the columns of `u` contain the values of the user functions.

The structure `s` contains information about the singular and user points that are located along the curve. `s` is stored in the `.mat` file and can be loaded using the standard MATLAB-function `load`. The elements of `s` are structures with the following fields:

`s(i).index` the index of the singular point in the `.dat` file.

`s(i).label` the standard CL_MATCONTL label for the located singularity, see Section 1.1.

`s(i).msg` a string describing the singularity

`s(i).data` a structure with more information about the singular point. The `data` structure always contains the fields

`data.x` contains the the state vector and values of the active parameters at the singular point.

`data.v` contains the tangent vector at the singular point

`data.P0` the value of the parameter vector `p` at the singular point

`data.ap` indices of the active parameters.

Further, data can contain information about the testfunctions, userfunctions, eigenvalues, normal form coefficients, and the matrices used in the CIS algorithm. Whether this information is available depends on the options used during continuation and the type of singularity that is located.

3.2.5 List of options

Here we give a list of the options supported by CL_MATCONTL. These options should be set using `contset`, see Section 3.2.1.

Cont_LogFile Boolean indicating whether a log file is generated in the `/Logs/` directory (default: 0)

Cont_DiagnosticsLevel Diagnostic level changes the amount of information written to screen and logfile. There are the following possible values

-inf: Suppress all output

0: Standard Output Information about singularities and general information about the curve will be displayed. Level 0 information is always output to the screen when using a log-file.

1: Additional Information Test functions values, step size.

2: Advanced information Additional information about the convergence of the location of singularities.

3: Maximal information The dimension of the invariant subspace and unstable subspace (CIS). Eigenvalues near zero.

4: Developer information Custom messages for development of new code. (Not intended for users)

5: Debugging Information function markers and other debugging information. (Not intended for users)

(default: 0).

Cont_Direction Boolean indicating the direction of the continuation (sign of the initial tangent vector) v_0 (default: 0)

Cont_InitStepsize The initial stepsize (default: 0.01)

Cont_MaxStepsize The maximum stepsize (default: 0.1)

Cont_MinStepsize The minimum stepsize (default: 10^{-5})

Cont_SmoothingAngle Maximum allowed angle between tangent vectors at adjacent points (default: $\frac{\pi}{50}$))

Cont_Stepsize_Inc_fac Factor with which the stepsize is increased after a new point on the curve has been found (default: 1.3)

Cont_Stepsize_Dec_fac Factor with which the stepsize is decreased when failing to find a new point on the curve (default: 0.5)

Cont_MaxNumPoints Maximum number of points on the curve (default: 300)

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

Cont_AdaptSteps Number of points indicating when to adapt the problem while computing the curve (default: 3)

Cont_MaxCorrIters Maximum number of correction iterations (default: 10)

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

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

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

Cont_Solver Name of the solver to be used during Newton iteration (default: "Moore-Penrose")

Cont_Eigenvalues Boolean indicating the computation of the eigenvalues (default: 0)

Cont_IncrFinDiff The increment to compute the derivatives numerically (default: 10^{-5})

Cont_ParallelComputing Boolean indicating whether the parallel computing toolbox should be used in the computation of the derivatives by finite differences. (default: 0)

Cont_Singularities Boolean indicating whether singularities should be detected and located along the curve (default: 0)

Cont_Userfunctions Boolean indicating whether roots of userfunctions are located (default: 0)

Cont_UserFuncInfo A cell array with information about the userfunctions. For every user function specified in the Problem definition file, the array must have an element with the following fields:

`.label`	short (typically 2 character) label of the userfunction
`.name`	name of this particular userfunction
`.state`	boolean indicating whether the userfunction is active

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

Loc_Testf_MaxIters Maximum number of iterations to locate a zero of a testfunction by bisection (default: 10)

Loc_Testf_FunTolerance Tolerance of test functions during bisection to locate a zero of a testfunction (default: 10^{-5})

Loc_Testf_VarTolerance Tolerance for variables during bisection to locate a zero of a testfunction (default: 10^{-4})

Loc_UseLocators Boolean vector indicating whether a Newton based locator should be used (when available) to locate the singularity. (default: empty)

Locator_MaxIters Maximum number of iterations to locate a singularity by Newton based method (default: 5)

Locator_FunTolerance Tolerance for function values in the Newton based method used to locate a singularity (default: 10^{-5})

Locator_VarTolerance Tolerance for variables in Newton based method used to locate singularity (default: 10^{-4})

Loc_Userf_MaxIters Maximum number of iterations to locate a zero of a userfunction by bisection (default: 10)

Loc_Userf_FunTolerance Tolerance of user functions during bisection to locate a zero of a userfunction (default: 10^{-5})

Loc_Userf_VarTolerance Tolerance for variables during bisection to locate a zero of a userfunction (default: 10^{-4})

EQ_BranchingMethod Method used for branch switching with the following possible values (see also Section 2.1.2)

0 Normal Form Method

1 Perpendicular Guess

2 Bisection

(default: 0)

EQ_SingLinSystTol Tolerance for singular linear system in equilibrium locators (default: 0.1)

CIS_usingCIS Boolean indicating whether the CIS algorithm is used (default: 1)

CIS_AdaptOnOverlap Boolean indicating whether the subspace is adapted on overlap (default: 1)

CIS_SparseSolvers Boolean indicating whether sparse solvers are used (default: 0)

CIS_InitNUnstable Number of initial unstable eigenvalues. A negative number forces contCIS to compute the number of unstable eigenvalues. (default: -1)

CIS_NUnstableGuess Initial guess for the number of unstable eigenvalues. (default: 15)

CIS_MaxUnstable Maximum number of the unstable eigenvalues. (default: 40)

CIS_InitSubSize Initial Subspace Size (default: 4)

CIS_NStableRef The number of stable Reference Eigenvalues for CIS (default: 4)

CIS_NExtra The number of extra eigenvalues that are computed to detect overlap. (default: 4)

RIC_Euler Boolean indicating whether Euler predictor in Ricatti solvers is used (default: 1)

RIC_MaxNewtonIters Maximum number of Newton steps in Ricatti solvers (default: 10)

RIC_FunTolerance Newton residual convergence tolerance in Ricatti solvers (default: $1e-5$)

RIC_VarTolerance Newton step convergence tolerance in Ricatti solvers (default: $1e-5$)

RIC_Transform Ricatti solver transformation ('none'/'invert'/'cayley') (default: 'none')

RIC_Cayley_Shift Cayley transform shift parameter in Ricatti solvers (default: 1)

RIC_Select Method used to select subspace in Ricatti solvers (default: 'ric')

RIC_Partial_Q Keep only Q1 even in dense case in ricatti solvers (default: 0)

TestPath The full path to the test run file. This is the directory in which the `/Logs/` and `/Data/` directories are created.
(default: `mfilename('fullpath')`)

Filename Name of the data, logs and singularities files in which continuation data of the computed curve is saved. When empty the name of the run script is used (with a time stamp to avoid overwriting previous runs).
(default: empty)

References

[1] A. D. Bazykin, Y. A. Kuznetsov, and A. I. Khibnik, *Portrety bifurkatsii*, vol. 89 of Новое в Жизни, Науке, Технике: Серия "Математика, Кибернетика" [Current Life, Science and Technology: Series "Mathematics and Cybernetics"], "Znanie", Moscow, 1989. Bifurkatsionnye diagrammy dinamicheskikh sistem na ploskosti. [Bifurcation diagrams of dynamical systems on the plane].

[2] D. Bindel, J. Demmel, and M. Friedman, *Continuation of invariant subspaces for large bifurcation problems*, in Proceedings of the SIAM Conference on Linear Algebra, Williamsburg, VA, 2003.

[3] ——, *Continuation of invariant subspaces for large bifurcation problems*, Tech. Report UCB/EECS-2006-13, EECS Department, University of California, Berkeley, February 13 2006.

[4] ——, *Continuation of invariant subspaces for large bifurcation problems*, SIAM J. Sci. Comput., 30 (2008), pp. 637–656.

[5] D. Bindel, W. Demmel, M. Friedman, W. Govaerts, and Yu.A. Kuznetsov, *Bifurcation analysis of large equilibrium systems in matlab*, in Proceedings of the ICCS conference 2005, volume 3514/2005, Atlanta, GA, 2005, pp. 50–57.

[6] D. Bindel, M. Friedman, W. Govaerts, J. Hughes, and Yu.A. Kuznetsov, *Numerical computation of bifurcations in large equilibrium systems in MATLAB*, 2014.

[7] D. Bindel, W. Govaerts, J. Hughes, Yu.A. Kuznetsov, M. Pekkér, and D. Veldman, *CL_MATCONTL Continuation Toolbox in MATLAB*, Oct. 2015. `http://uah.edu/faculty/pekker`.

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

[9] A. Dhooge, W. Govaerts, and Yu.A. Kuznetsov, matcont*: A* matlab *package for numerical bifurcation analysis of odes.*, ACM TOMS., 29 (2003), pp. 141–164.

[10] A. Dhooge, W. Govaerts, Yu.A. Kuznetsov, W. Mestrom, and A. M. Riet, *MATLAB continuation software package CL_MATCONT*, Dec. 2008. `http://sourceforge.net/projects/matcont/`.

[11] L. Dieci and M. J. Friedman, *Continuation of invariant subspaces*, Numerical Linear Algebra and Appl., 8 (2001), pp. 317–327.

[12] F. Dumortier, S. Ibáñez, and H. Kokubu, *New aspects in the unfolding of the nilpotent singularity of codimension three*, Dyn. Syst., 16 (2001), pp. 63–95.

[13] F. Dumortier, R. Roussarie, J. Sotomayor, and H. Żołpolhk adek, *Bifurcations of planar vector fields*, vol. 1480 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1991. Nilpotent singularities and Abelian integrals.

[14] E. Freire, E. Gamero, A. J. Rodríguez-Luis, and A. Algaba, *A note on the triple-zero linear degeneracy: normal forms, dynamical and bifurcation behaviors of an unfolding*, Internat. J. Bifur. Chaos Appl. Sci. Engrg., 12 (2002), pp. 2799–2820.

[15] M. Friedman, *Improved detection of bifurcations in large nonlinear systems via the Continuation of Invariant Subspaces algorithm*, Int. J. Bif. and Chaos, 11 (2001), pp. 2277–2285.

[16] M. Friedman, W. Govaerts, Yu.A. Kuznetsov, and B. Sautois, *Continuation of homoclinic orbits in MATLAB*, in Proceedings of the ICCS conference 2005, volume 3514/2005, Atlanta, GA, 2005, pp. 263–270.

[17] M. Friedman and W. Qiu, *On the location and continuation of Hopf bifurcations in large-scale problems*, Int. J. Bif. and Chaos, 18 (2008), pp. 1589–1597.

[18] W. Govaerts, *Numerical methods for bifurcations of dynamical equilibria*, SIAM, Philadelphia, 2000.

[19] J. Hughes and M. Friedman, *A bisection-like algorithm for branch switching at a simple branch point*, J. Sci. Comput., 41 (2009), pp. 62–69.

[20] Yu. A. Kuznetsov, *Elements of Applied Bifurcation Theory, Third edition*, Springer-Verlag, New York, 2004.

[21] Yu.A. Kuznetsov, *Practical computation of normal forms on center manifolds at degenerate Bogdanov–Takens bifurcations*, International Journal of Bifurcation and Chaos, 15 (2005), pp. 3535–3546.

[22] W. Weymiens, H. de Blank, and G. Hogeweij, *Bifurcation theory of a one-dimensional transport model for the L–H transition*, Physics of Plasmas, 20 (2013), p. 082306.

[23] W. Weymiens, H. de Blank, G. Hogeweij, and J. de Valença, *Bifurcation theory for the L–H transition in magnetically confined fusion plasmas*, Physics of Plasmas, 19 (2012), p. 072309.

[24] W. Weymiens, S. Paquay, H. de Blank, and G. Hogeweij, *Comparison of bifurcation dynamics of turbulent transport models for the L–H transition*, Physics of Plasmas, 21 (2014), p. 052302.

[25] H. Zohm, *Dynamic behavior of the L–H transition*, Physical review letters, 72 (1994), p. 222.