PyCont: Continuation Types

Contents

1 Introduction and notation

Consider the following differential equation:

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{F}(\boldsymbol{y}, \boldsymbol{a}),$$

where $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ are phase variables, $\boldsymbol{a} = (a_1, a_2, \ldots, a_m)$ are parameters, and $\boldsymbol{F} = (F_1(\boldsymbol{y}, \boldsymbol{a}), \ldots, F_n(\boldsymbol{y}, \boldsymbol{a}))$ are n functions. The jacobian of $\boldsymbol{F}$ will be denoted by $\boldsymbol{F}_{\boldsymbol{y}}$.

PyCont: The function $\boldsymbol{F}$ is stored in the variable `self.sysfunc`.

2 Bordered Matrix Methods (`class BorderMethod(TestFunc)`)

Suppose we have a test function that signals a bifurcation point when $\det(A) = 0$, where A is an $n \times n$ matrix. We can consider the bordered extension M of A given by

$$M = \begin{pmatrix} A & b \\ c^T & d \end{pmatrix},$$

where $b, c \in \mathbb{R}^n$ and $d \in \mathbb{R}$. If we suppose that M is nonsingular and we solve the system

$$M \begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} 0_n \\ 1 \end{pmatrix},$$

where $r \in \mathbb{R}^n$ and $s \in \mathbb{R}$, then by Cramer's rule we have

$$s = \frac{\det(A)}{\det(M)}.$$

Thus, $\det(A) = 0$ if and only if $s(A) = 0$.

In the general case, we have

$$\begin{pmatrix} A & B \\ C^T & D \end{pmatrix} \begin{pmatrix} V \\ G \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{1}$$

where

$$\begin{aligned} r &= \text{corank} \\ A &= n \times m \\ s &= \max(n, m) \\ p &= s - m + r \\ q &= s - n + r \\ B &= n \times p \\ C &= m \times q \\ D &= 0_{q \times p} \\ V &= m \times q \\ G &= p \times q \\ 0 &= n \times q \\ 1 &= q \times q. \end{aligned}$$

The matrix G can also be seen as a function of A as

$$G_{\text{bor}} : \mathbb{R}^{nm} \to \mathbb{R}^{pq} \qquad \text{such that} \qquad G_{\text{bor}}(A) = G.$$

It can also be written as

$$\begin{pmatrix} W^T & G \end{pmatrix} \begin{pmatrix} A & B \\ C^T & D \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix}, \tag{2}$$

where now

$$\begin{aligned} W &= n \times p \\ 0 &= p \times m \\ 1 &= p \times p. \end{aligned}$$

This is important for calculating derivatives, since we have

$$G_z = -W^T A_z V.$$

2.1 Initialization (`BorderMethod.setdata`)

In the bordered matrix methods, we need to initialize the B and C matrices so that the matrix M is nonsingular. Suppose we have the singular value decomposition of A as $A = U\Sigma Z^T$, where U is $n \times t$, Σ is $t \times t$, Z is $m \times t$ and $t = \min(n, m)$. We initialize B and C as follows:

$$\begin{aligned} B &= \begin{pmatrix} U_{t-p+1} & \cdots & U_t \end{pmatrix} \\ C &= \begin{pmatrix} Z_{t-q+1} & \cdots & Z_t \end{pmatrix} \end{aligned}$$

Note that $p, q \leq t$.

2.2 Function evaluation (`BorderMethod.func`)

By using the LU factorization of M, we can solve (1) and (2) for V, W and G. We then update the matrices B and C as follows:

$$\begin{aligned} B &= ||A||_1 \frac{W}{||W||_1}, \\ C &= ||A||_\infty \frac{V}{||V||_1}. \end{aligned}$$

3 Codimension 1

3.1 Continuous Dynamical Systems

3.1.1 Equilibrium Curves (EP-C) (`class EquilibriumCurve(Continuation)`)

3.1.1.1 Mathematical definition In this case, we are concerned with curves of *equilibrium points* (EP) as a function of a free parameter a_1, defined by

$$\boldsymbol{F}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{a}}) = \boldsymbol{0},$$

where $\boldsymbol{F} : \mathbb{R}^{n+1} \to \mathbb{R}^n$, $\tilde{\boldsymbol{y}} = (y_1, \ldots, y_n, a_1)$ and $\tilde{\boldsymbol{a}} = (a_2, \ldots, a_m)$.

The phase variables $(y_1, \ldots, y_n)$ are stored in `self.coords`, while the free parameter a_1 is stored in `self.params`.

The jacobian is given by $\boldsymbol{F}_{\tilde{\boldsymbol{y}}} : \mathbb{R}^{n+1} \to \mathbb{R}^n$, where

$$\boldsymbol{F}_{\tilde{\boldsymbol{y}}}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{a}}) = \left(\boldsymbol{F}_{\boldsymbol{y}} \ \boldsymbol{F}_{a_1}\right).$$

3.1.1.2 Detection of bifurcation points We have the following bifurcation points on an equilibrium curve:

- Branch Bifurcation Point (BP) (`class BranchPoint(BifPoint)`)
- Fold Bifurcation Point (LP) (`class FoldPoint(BifPoint)`)
- Hopf Bifurcation Point (H) (`class HopfPoint(BifPoint)`)

To detect these bifurcation points, we use the following test functions:

$$\phi_1(\tilde{\boldsymbol{y}}) = \det \begin{pmatrix} \boldsymbol{F}_{\tilde{\boldsymbol{y}}} \\ \boldsymbol{V}^T \end{pmatrix} \qquad (\texttt{Branch\_Det}) \tag{3}$$

$$\phi_2(\tilde{\boldsymbol{y}}) = V_{n+1} \qquad (\texttt{Fold\_Tan}) \tag{4}$$

$$\phi_3(\tilde{\boldsymbol{y}}) = G_{\text{bor}}(2\boldsymbol{F}_{\boldsymbol{y}} \odot I_n) \qquad (\texttt{Hopf\_Bor}) \tag{5}$$

	ϕ_1	ϕ_2	ϕ_3
BP	0	-	-
LP	1	0	-
H	-	-	0

In the table above, a zero and a one corresponds to the test functions being zero or nonzero, respectively. Alternate test functions include:

$$\phi(\tilde{\boldsymbol{y}}) = \det(\boldsymbol{F}_{\boldsymbol{y}}) \qquad (\texttt{Fold\_Det})$$

$$\phi(\tilde{\boldsymbol{y}}) = G_{\text{bor}}(\boldsymbol{F}_{\boldsymbol{y}}) \qquad (\texttt{Fold\_Bor})$$

$$\phi(\tilde{\boldsymbol{y}}) = \det(2\boldsymbol{F}_{\boldsymbol{y}} \odot I_n) \qquad (\texttt{Hopf\_Det})$$

$$\textit{NOT USED} \qquad (\texttt{Hopf\_Eig})$$

3.1.1.3 Location of bifurcation points (general)

Algorithm: `Locate zeros of test functions`

Input: `Two points on the curve given by` (x_1, v_1) `and` (x_2, v_2) `such that` $\phi_1(x_1, v_1) < 0$ `and` $\phi_1(x_2, v_2) > 0$

Output: `Found point` (x, v)

$\Phi_1 := \phi_1(x_1, v_1)$
$\Phi_2 := \phi_1(x_2, v_2)$
for $i :=$ `1 to MaxTestIters`
    $r := \left|\frac{\Phi_1}{\Phi_1 - \Phi_2}\right|$
    if $r \geq 1$
        $r = 0.5$
    $x := x_1 + r(x_2 - x_1)$
    $v := v_1 + r(v_2 - v_1)$
    $(x, v) :=$ `Corrector`$\big((x, v)\big)$
    $\Phi := \phi_1(x, v)$
    if $|T| <$ `TestTol` **and** $\min(|x - x_1|, |x - x_2|) <$ `VarTol`
        break
    else
        if $\text{sign}(\Phi) == \text{sign}(\Phi_2)$
            $(x_2, v_2, \Phi_2) := (x, v, \Phi)$
        else
            $(x_1, v_1, \Phi_1) := (x, v, \Phi)$
return (x, v)

3.1.1.4 Location of branch points (`class BranchPoint(BifPoint).locate`) As mentioned in MATCONT, the region of attraction near a BP point has the shape of a cone, which we cannot guarentee to stay within. We thus define temporary variables $\beta \in \mathbb{R}$ and $p \in \mathbb{R}^n$ and implement Newton's method in the space $(\tilde{\boldsymbol{y}}, \beta, p) \in \mathbb{R}^{2(n+1)}$ with the extended system given by:

$$\left\{ \begin{array}{rcl} \boldsymbol{F}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{a}}) + \beta p & = & 0 \\ \left[\boldsymbol{F}_{\boldsymbol{y}}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{a}}) \ \boldsymbol{F}_{a_1}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{a}})\right]^T p & = & 0 \\ p^T p - 1 & = & 0 \end{array} \right. \tag{6}$$

We start with $\beta = 0$ and p the left eigenvector of $\boldsymbol{F}_{\boldsymbol{y}}$ associated with the smallest eigenvalue.

3.1.1.4.1 Computation of branch direction NOTE: Doesn't currently work!!!!! See [1] for mathematical discussion, notes in NoteTakerHD and PyCont_Brusselator.py for example code.

Setting ψ to the p found upon convergence in the above Newton's method, we first set V_1 to the real part of the eigenvector associated with the smallest (i.e. zero) eigenvalue of the matrix associated with the test function (3)

$$\begin{pmatrix} \boldsymbol{F}_{\tilde{\boldsymbol{y}}} \\ \boldsymbol{V}^T \end{pmatrix}.$$

Then, given the Hessian H of $\boldsymbol{F}(\tilde{\boldsymbol{y}})$, we compute the following scalars:

$$\begin{aligned}
c_{11} &= \psi^T H[V, V] \\
c_{12} &= \psi^T H[V, V_1] \\
c_{22} &= \psi^T H[V_1, V_1] \\
\beta &= 1 \\
\alpha &= -\frac{c_{22}}{2c_{12}}
\end{aligned}$$

We then compute the direction of the new branch as

$$V_{\text{new}} = \alpha V + \beta V_1.$$

Note: c_{11} is not used in the computation but is included for completeness. Also, $\beta = 1$ and so can be omitted. (I need to find reference for this!)

3.2 Discrete Dynamical Systems

4 Codimension 2

4.1 Continuous Dynamical Systems

4.1.1 Fold Curves (LP-C) (`class FoldCurve(Continuation)`)

In this case, we are concerned with curves of *fold bifurcation points* (LP) as a function of two free-parameters (a_1, a_2), defined by the augmented system

$$\boldsymbol{C}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{a}}) = \left\{ \begin{array}{c} \boldsymbol{F}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{a}}) \\ G_{\text{bor}}(\boldsymbol{F}_{\boldsymbol{y}}) \end{array} \right. = \boldsymbol{0}, \tag{7}$$

such that $\boldsymbol{C} : \mathbb{R}^{n+2} \to \mathbb{R}^{n+1}$, $\tilde{\boldsymbol{y}} = (y_1, \ldots, y_n, a_1, a_2)$ and $\tilde{\boldsymbol{a}} = (a_3, \ldots, a_m)$. We have the following bifurcation points on a fold curve:

- Bogdanov-Takens (BT) (`class BTPoint(BifPoint)`)
- Zero-Hopf point (ZH) (`class ZHPoint(BifPoint)`)
- Cusp point (CP) (`class CPPoint(BifPoint)`)
- Branch point (BP) (`class BranchPoint(BifPoint)`)

For the bordered method `Fold_Bor` in (7), we have $p = q = 1$, and thus the vectors $v = V$ and $w = W$ in equations (1) and (2) are both $n \times 1$. They are updated continuously throughout the continuation, and are used in the test functions for these bifurcation points as follows:

$$\begin{aligned}
\phi_1(\tilde{\boldsymbol{y}}) &= w^T v && (\texttt{BT_Fold}) \qquad (8)\\
\phi_2(\tilde{\boldsymbol{y}}) &= G_{\text{bor}}(2\boldsymbol{F}_{\boldsymbol{y}} \odot I_n) && (\texttt{Hopf_Bor}, (5))\\
\phi_3(\tilde{\boldsymbol{y}}) &= w^T \boldsymbol{F}_{\boldsymbol{yy}}[v, v] && (\texttt{CP_Fold}) \qquad (9)\\
\phi_4(\tilde{\boldsymbol{y}}) &= w^T [\boldsymbol{F}_{a_1} \; \boldsymbol{F}_{a_2}] && (\texttt{BP_Fold}) \qquad (10)
\end{aligned}$$

	ϕ_1	ϕ_2	ϕ_3	$\phi_{4,1}$	$\phi_{4,2}$
BT	0	0	-	-	-
ZH	1	0	-	-	-
CP	-	-	0	-	-
BP	-	-	-	0	-
BP	-	-	-	-	0

For an example of branch points on a fold curve, see PyCont_BranchFold.py.

References

[1] Wolf-Jürgen Beyn, Alan Champneys, Eusebius Doedel, Willy Govaerts, Yuri A Kuznetsov, and Björn Sandstede. Numerical Continuation, And Computation Of Normal Forms. In *In Handbook of dynamical systems III: Towards applications.*