Fold and branch point continuation in a Schnakenberg system and details of branch plotting – a `pde2path` tutorial

Hannes de Witt[1]

[1] Institut für Mathematik, Universität Oldenburg, D26111 Oldenburg, hannes.de.witt@uni-oldenburg.de

April 4, 2017

Abstract

We describe the `OOPDE` settings in `pde2path` for a Schnakenberg system with fold and branch point continuation in 1D and 2D. Additionally details of the `pde2path` function `plotbra`, which plots and labels branches, are discussed.

1 Introduction

The aim of this tutorial is to extend [RU17] to the system case. As a model system we consider the modified Schnakenberg system

$$0 = D\Delta U + F(u), \quad F(u) = \begin{pmatrix} -u+u^2v \\ \lambda - u^2v \end{pmatrix} + \sigma \left(u - \frac{1}{v}\right)^2 \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{1}$$

with $U = (u, v)(x) \in \mathbb{R}^2$, $x \in \Omega \subset \mathbb{R}^n$, $n = 1, 2$, Ω an interval or a rectangle and diffusion matrix $D = \left(\begin{smallmatrix} 1 & 0 \\ 0 & d \end{smallmatrix}\right)$. Excepting §6 we stick to $d = 60$ where analytically a bifurcation of a periodic solution at $\lambda_c = \sqrt{60}\sqrt{3-\sqrt{8}} \approx 3.21$ with critical wave number $k_c = \sqrt{2}\sqrt{\sqrt{2}-1}$ occurs. Details on the theory of this system can be found in [UW14]. In general the continuation is more complicated in (1) than in the Allen-Cahn models discussed in [RU17], as there are many more solution branches, also see [UWR14, §4.2] for further comments. However, in this tutorial we avoid these problems by considering relatively simple domains and branches.

`pde2path` uses a problem structure, which we call `p`, to store the data. In particular the unknown function $U = (u, v) = (u_1, u_2)$ is stored in `p.u` along with the parameters λ, σ and d. The parameters must always be placed as the last entries of `p.u`. In detail, `p.u` will be the vector $(u_1, u_2, \lambda, \sigma, d)$ in this tutorial. As `pde2path` stores as many solutions as desired on hard disk it is suggested to work with `p` as the only problem structure and load old solutions with `loadp` if needed, even though one could define arbitrary many problem structure variables. See also [dWDR+17] for a data structure overview and quick references.

The tutorial is organized as follows. In §2 we explain the basic implementation of the Schnakenberg model in `pde2path` and explain the general settings. In §3 we will use a basic set of commands to numerically compute the trivial and the periodic branch bifurcating at λ_c along with a first fold continuation. §4 works as an exercise and gives a more detailed bifurcation diagram with some hints to recreate this. §5 takes a look at the model in a two dimensional domain and shows fold continuation there. In §6 we will see the continuation of a branch point instead of a fold point in 1D. In §7 we will give some details of the branch plotting function `plotbra` and show several example plots.

In general the output generated by `plotbra` profits from some post processing with the standard `Matlab` tools, but as these shall not be discussed here, we restrict it to a minimum.

The files corresponding to this tutorial are located in the demo folder `schnakfold`. Table 1 gives an overview. As this tutorial does not focus on solution plots, we outsourced these to the file `cmds_sp.m`. For detailed informations about solution plots see [Wet17].

Table 1: File structure of the demo `schnakfold`.

file	description
`sG.m`	residual of the pde
`sGjac.m`	Jacobian
`spjac.m`	Jacobian for spectral continuation
`oosetfemops.m`	generates `FEM`/mass-matrices
`schnakinit.m`	initialization of the problem
`schnakbra_f1D.m`	plotting function, used in §5 only
`cmds_f1D.m`	fold continuation in 1D
`cmds_f2D.m`	fold continuation in 2D
`cmds_b1D.m`	branch continuation in 1D
`cmds_ex.m`	solution for the 1D exercise in §4
`cmds_plot.m`	several plot commands
`cmds_sp.m`	solution plots

2 Basic setup

In the `OOPDE` setting we use here, see [Prü16] for detailed background and [RU17] for an introduction, the basic setup of each `pde2path` problem consists of at least three parts. The `oosetfemops.m` function to compute the needed `FEM` matrices, the `sG` and `sGjac` functions for the residual of the PDE respectively the (analytical) Jacobian, and an init command file or function to set some basic parameters. We now discuss these three parts in detail, e.g. give the code and some remarks.

Let us take a look at the implementation of the model functions first. Listing 1 shows the implementation of the residual. We use some matrices `p.mat.K` and `p.mat.M` here which are the `FEM` and mass matrices and will be defined in `oosetfemops.m`, see Listing 3. While it is convenient to store the full mass matrix in `p.mat.M` it is sufficient to store the diffusion matrix for a scalar problem in `p.mat.K` only and assemble the full matrix in `sG.m` respectively `sGjac.m`. For the implementation of arbitrary diffusion coefficients, like d here, it even is necessary.

```
function r=sG(p,u)
    u1=u(1:p.np); % solution component 1
    u2=u(p.np+1:2*p.np); % solution component 2
    par=u(p.nu+1:end); % parameters
    f1=-u1+u1.^2.*u2+par(2)*(u1-u2.^(-1)).^2; % F_1(u), see eqn (1) in tut
    f2=par(1)-u1.^2.*u2-par(2)*(u1-u2.^(-1)).^2; % F_2(u)
    f=[f1;f2];
    K=kron([[1,0];[0,par(3)]],p.mat.K); % assemble full FEM matrix
    r=K*[u1;u2]-p.mat.M*f; % the residual
end
```

Listing 1: sG.m; calculation of the residual in `FEM`-discretization. `p.u` is the vector $(u_1, u_2, \lambda, \sigma, d)$, i.e. par$= (\lambda, \sigma, d)$.

The Jacobian is created in the same way, see Listing 2.

```
function Gu=sGjac(p,u)
    par=u(p.nu+1:end); % parameters
    n=p.np;
    [f1u,f1v,f2u,f2v]=njac(p,u,par); % the Jacobian, see below
    Fu=[[spdiags(f1u,0,n,n),spdiags(f1v,0,n,n)];
       [spdiags(f2u,0,n,n),spdiags(f2v,0,n,n)]];
    Gu=kron([[1,0];[0,par(3)]],p.mat.K)-p.mat.M*Fu; % assemble the Jacobian
end
function [f1u,f1v,f2u,f2v]=njac(p,u,par) % Jacobian for Schnakenberg
    u1=u(1:p.np); % solution component 1
    u2=u(p.np+1:2*p.np); % solution component 2
    % entries of the jacobian
```

```
    f1u=-1+2*u1.*u2+2*par(2)*(u1-u2.^(-1));
    f1v=u1.^2+2*par(2)*u2.^(-2).*(u1-u2.^(-1));
    f2u=-2*u1.*u2-2*par(2)*(u1-u2.^(-1));
    f2v=-u1.^2-2*par(2)*u2.^(-2).*(u1-u2.^(-1));
end
```

Listing 2: sGjac.m; calculation of the Jacobian in FEM-discretization.

When coding `sG.m` and `sGjac.m` it is advised to separate the components of the unknown function and the parameters as done here. Except `p.mat.K` and `p.mat.M`, the model is completely implemented now and these two are initialized by `oosetfemops.m`, see Listing 3.

```
function p=oosetfemops(p)
[p.mat.K,M,~]=p.pdeo.fem.assema(p.pdeo.grid,1,1,1); % FEM/mass matrices
% mass matrix adaption for the problem, as it is a system
p.mat.M=kron([[1,0];[0,1]],M);
end
```

Listing 3: oosetfemops.m; as here we use homogeneous Neumann BC we only need to call `assema` and all boundary terms are zero. The stiffness matrix K and the mass matrix M are then saved in `p.mat.K` and `p.mat.M`.

If one hard codes the diffusion parameter, e.g. implements it with an explicit value, one can also initialize the whole diffusion matrix in `oosetfemops.m`.

With these functions the Schnakenberg system is fully implemented, and to start a continuation only three things miss: the domain, a set of parameters $(\lambda_0, \sigma_0, d_0)$ where the continuation shall start and a rough guess for a solution at this setting. It is convenient to put some initializations into an init function. Listing 4 gives basic settings for this example.

```
function p=schnakinit(p,dom,mp,par)
    %% setting standard parameters
    p=stanparam(p); % infuses p with standard parameter settings
    screenlayout(p); % open, clear and arrange the common figures

    %% special parameters related to this model
    % basics
    p.nc.neq=2; % number of equations in the model
    p.sw.sfem=-1; % type of numerical calculation, here OOPDE
    p.sw.spjac=1; % use analytical Jacobian for spectral point cont (fold cont)
    % names for cmp of stanbra
    p.plot.auxdict={'\lambda','\sigma','d','||u_1||_{\infty}','min(|u_1|)'};
    % description of the model
    p.fuha.sG=@sG; % the model itself
    p.fuha.sGjac=@sGjac; % the Jacobian of the model
    p.fuha.spjac=@spjac; % Jacobian for spectral point cont (fold cont)

    %% domain and mesh
    kc=sqrt(sqrt(2)-1); % wavenumer of the critical mode
    switch length(dom);
        case 1;
            lx=dom*2*pi/kc; % set domain length according to critical wavenumber
            p.pdeo=stanpdeo1D(lx,2*lx/mp); % mesh [-lx,lx], max mesh pt 2*lx/r
        case 2;
            nx=dom(1)*mp;
            ny=dom(2)*mp;
            lx=dom(1)*2*pi/kc; % set domain x-length
            ly=dom(2)*2*pi/sqrt(3)/kc; % set domain x-length
            p.pdeo=stanpdeo2D(lx,ly,nx,ny); % mesh [-lx,lx]x[-ly,ly] mesh pt nx*
   ny
    end
    p.np=p.pdeo.grid.nPoints; % number of meshpoints
```

```
    p.nu=p.np*p.nc.neq; % number of unknowns (=2*(mesh points), as 2 components)
    p=setfemops(p); % compute FEM-operators

    %% bifurcation parameter, continuation basics and first guess for solution
    p.nc.ilam=1; % primary bifurcation parameter located at p.u(p.np+p.nc.ilam)
    p.sol.xi=1/p.nu; % weight in arclength-continuation
    p.sol.ds=-0.01; % starting stepsize
    p.nc.dsmax=0.01; % maximal stepsize
    p.nc.dsmin=0; % minimal stepsize
    % construction the trivial solution
    lam=par(1); % setting parameter lambda
    u=lam*ones(p.np,1); % initial guess for u resp. u_1
    v=(1/lam)*ones(p.np,1); % initial guess for v resp. u_2
    p.u=[u;v;par']; % initial solution guess with parameters
end
```

Listing 4: schnakinit.m; initialization file. Sets basic informations for the mesh/domain, the solution, some parameters, and makes the model information stored in `sG.m`, `sGjac.m` and `oosetfemops.m` accessible to `pde2path`. The initial solution, lines 42ff, is explicit known here, but a rough guess is sufficient, see [RU17, §3.1.3] for an example.

There are many additional options, and when working with a problem one may have to change some of these, for example the step size ds. See [dWDR+17] for details. It is generally useful to give the init function some inputs for domain, number of mesh points, value of parameters and so on to be able to use the file for several investigations of the problem. Here we restricted the inputs to a domain parameter *dom*, the number of mesh points *mp* and the values of the parameter organized in a vector $[\lambda, \sigma, d]$. The switch distinguishes 1D and 2D by the type of the domain input. 2D will be discussed later, for now only case one is relevant.

The function `spjac.m`, set in the init file at line 15 is the analytical Jacobian for spectral continuation and is used for fold and branch point continuation. It is not necessary to define it, but, as it has to be calculated numerically otherwise, speeds up fold continuation a lot. The corresponding `spjac.m` file is shown in Listing 5. To check the calculations in `sGjac.m` and `spjac.m`, call `jaccheck(p)` and `spjaccheck(p)` which compares the implemented derivatives of `sG` and `sGjac` with a finite difference approximation. While `jaccheck` can be used directly after initializing the problem, `spjaccheck` has to be used after the call of `spcontini`, which initialize the fold respectively branch point continuation. See §3 for details.

```
function Gvvph=spjac(p,u)
u1=u(1:p.np); % first component
u2=u(p.np+1:2*p.np); % second component
par=u(2*p.nu+1:end); % parameters
s=par(2); % sigma
n=p.np; % number of function points per component
ov=ones(n,1); % dummy for the 1 function
% second order derivations of the model
f1uu=2*u2+2*s*ov;
f1uv=2*u1+2*s*u2.^(-2);
f1vv=-4*s*(u1-u2.^(-1)).*u2.^(-3)+2*s*u2.^(-4);
f2uu=-f1uu;
f2uv=-f1uv;
f2vv=-f1vv;
% implementation of the derivations as sparse matrices
ph1=u(p.nu+1:p.nu+p.np);
ph2=u(p.nu+p.np+1:2*p.nu);
M1=spdiags(f1uu.*ph1+f1uv.*ph2,0,n,n);
M2=spdiags(f1uv.*ph1+f1vv.*ph2,0,n,n);
M3=spdiags(f2uu.*ph1+f2uv.*ph2,0,n,n);
M4=spdiags(f2uv.*ph1+f2vv.*ph2,0,n,n);
```

```
Gvvph=-p.mat.M*[[M1 M2]; [M3 M4]];
end
```

Listing 5: spjac.m; analytical Jacobian for spectral continuation used for fold continuation. Can be omitted if one sets `p.sw.spjac=0` but speeds up fold continuation as otherwise being calculated numerically.

3 Basic fold continuation

Now that we have the basic setting we can start the continuation. One can do this in the command window, but most of the time it is more useful to write a script file, for instance called `cmds.m`. Before starting with a new problem one wants to clear the workspace and close all plots. Instead of using the `Matlab` function `clear all` to clear the workspace we use `keep pphome`, as this will clear all entries besides `pphome` which is needed for some secondary functions of `pde2path`, in particular the help system. After clearing the workspace one initializes the problem variable p and fills it with the standard settings. Listing 6 lists the commands for a fold continuation on the periodic branch. It is highly recommended to run the script cell by cell.

```
%% 1 - creating a clear working space
close all; keep pphome;
%% 2 - initialising the problem
p=[];
par=[sqrt(60)*sqrt(3-sqrt(8))+5e-2, -0.6, 60]; % [lambda, sigma, d]
p=schnakinit(p,4,300,par);
p.plot.pmod=10; % shows each 10th solution in fig 2 only
p.file.smod=10; % stores each 10th solution only
p=setfn(p,'tr_f1D');
%% 3 - contiunation of the trivial branch
p=cont(p,20); % continuation for a maximum of 20 steps
%% 4 - switch to periodic branch and continuation with fold detection
p=swibra('tr_f1D','bpt1','per_f1D',1e-2); % switch to new branch with ds=0.01
p.sw.foldcheck=1; % enables detection of folds
p.sw.bifcheck=0; % disable detection of bifs
p=cont(p,150); % continuation for a maximum of 150 steps
%% 5 - fold continuation in sigma
p=spcontini('per_f1D','fpt1',2,'fold_f1D'); % switch to fold cont in par. sigma
p.sol.ds=-1e-3; % continue backward in sigma
clf(2);
p.plot.bpcmp=1; % plot lambda position of fold over sigma in fig 2 now
p=cont(p,50); % continuation for a maximum of 50 steps
%% 6 - continuation at new fold point in lam again
p=spcontexit('fold_f1D','pt50','per2a_f1D'); % exit fold continuation
p.sol.ds=1e-2; % continue forward in lambda
clf(2);
p.plot.bpcmp=0; % plot L2-norm over lambda in fig 2
p=cont(p,20); % cont for a maximum of 20 steps
p=spcontexit('fold_f1D','pt50','per2b_f1D'); % exit fold continuation
p.sol.ds=-1e-2; % continue forward in lambda
p.nc.lammin=3.2084; % minimal lambda - approx. bif point from trivial branch
p.plot.bpcmp=0; % plot L2-norm over lambda in fig 2
p=cont(p,120); % cont for a maximum of 120 steps
%% 7 - plot BD
figure(3);
clf;
plotbra('tr_f1D'); % trivial branch
plotbra('per_f1D'); % periodic nranch for sigma=-0.6
plotbra('per2a_f1D','cl','r'); % periodic branch for sigma=-0.7617
plotbra('per2b_f1D','cl','r'); % periodic branch for sigma=-0.7617
%% 8 - plot lambda over sigma for fold
```

```
figure(4);
clf;
plotbra('fold_f1D',4,1); % plot lambda position of fold over sigma
```

Listing 6: cmds_f1D.m; we continue the trivial branch, switch to a periodic one and make a fold continuation in the parameter σ there. The results are plotted.

Cells one and two initialize the problem. There should be three figures after the run of these. `Matlab`-figure 1 will show the current solution plot, `Matlab`-figure 2 will show a basic bifurcation diagram, and `Matlab`-figure 6 will show special plots like the tangent vector at bifurcation points if one switches the branch. Figure 1 and 2 should fill with life through the run of cell three which calculates a small part of the trivial branch. One should also see some output in the command window then. `Matlab`-figure 6 then fills by the use of `swibra` in cell four. To see what the further cells do, see the commented script in Listing 6. At cells seven and eight one should get the graphs shown in Figure 1a,b.

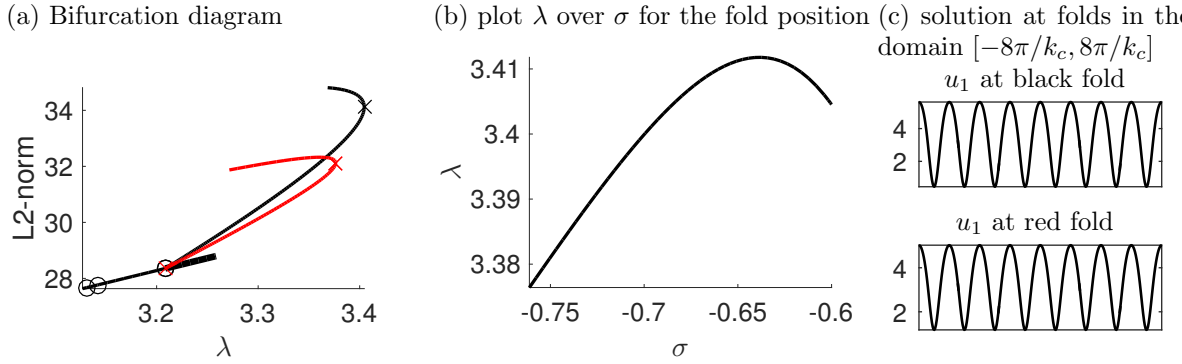


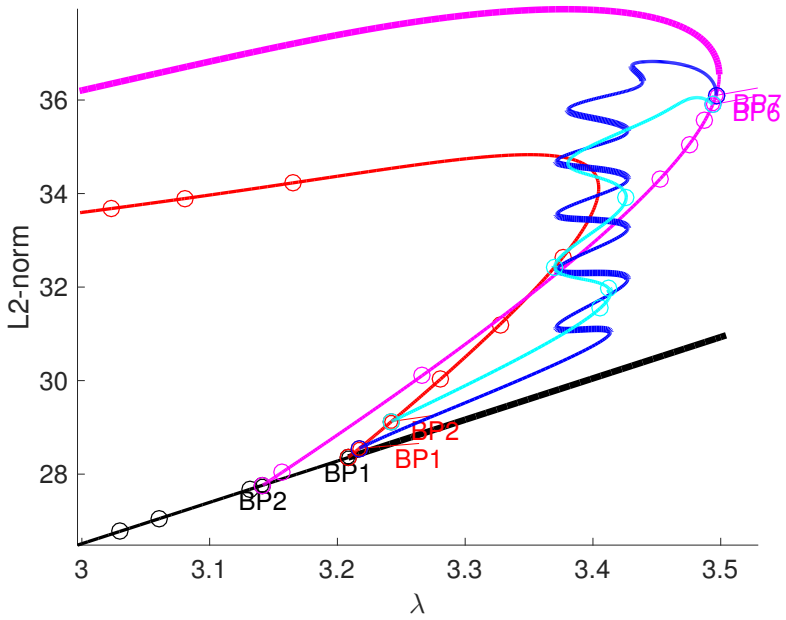
Figure 1: (a),(b) `Matlab` output after the run of `cmds_f1D.m`. (c) Solution plots at the fold of the red and black branch of periodic solutions. For details on the creation of the solution plots see [Wet17].

You may now modify the script on your own. But, as already mentioned in the introduction, the system (1) has many solutions even in 1D, so in particular a change of the domain will change the continuation results a lot, as additional bifurcation points will rise, and some might not be found with `cont` anymore, if to many bifurcation points are close to each other. This implies, that a change from 1D to 2D is – even though one only has to give another domain parameter – a delicate problem for this system. For this reason it is cumbersome to transfer the above example in 2D. That is why we continue with an 1D exercise, and will give another example for a 2D fold continuation afterwards.

4 Exercise 1D

Starting from the cmds file in the previous section, it is a good exercise to recreate the bifurcation diagram shown in Figure 2a and the fold continuation shown in Figure 3a. To do so it is advised to copy the files for the basic fold continuation in a new directory and modify them as necessary. The only numerical constants one wants to adjust for this plots are `p.nc.lammin` and `p.nc.lammax`. No changes in domain size or mesh points are needed. Also note, that the classical way to find the magenta branch would be to switch branch via `swibra` from the end of the snaking branch and continue in both directions, but this tends to fail. There are other ways. As stated in the caption, Figure 3a is created by following the third and forth fold point of the blue branch of localized patterns. See Figure 3b for a location of these. Possible solutions to create the plots are located in `cmds_ex.m`. Keep in mind, that multiple calls of the same plot will lead to slightly different figures,

(a) Bifurcation diagram ($\sigma = -0.6$)



(b) several solution plots in the domain $[-8\pi/k_c, 8\pi/k_c]$

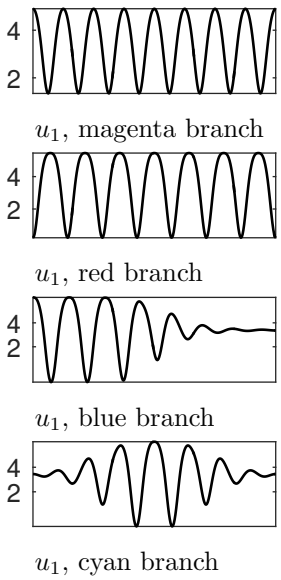


Figure 2: (a) detailed bifurcation diagram of the 1D Schnakenberg model for $\sigma = -0.6$. The labeling can be improved by various options, for instance through the use of `'fancy',2` as additional input for `plotbra`. See §7 for some details. (b) several solution plots generated with `plotsol`. See [Wet17] for a detailed description of `plotsol`.

as the labels offset is randomized. Thus a solution to the exercise might not exactly look like Figure 2a. If the verification of a solution is difficult one can plot the solution in `cmds_ex` without labels and compare this with a unlabeled version of the own solution. See §7 or [dWDR+17] for details on how to do so.

5 2D fold continuation

The init file for 2D has already been discussed in §2. The only difference lies in the call of `schnakinit` with a vector as the domain size, which will switch on case two and thus uses `stanpdeo2D` instead of `stanpdeo1D`. Additional changes which are commonly placed in the init file have been moved to lines 11-18 in the cmds file. These enhance the branch and solution plotting. As indicated in §3 even though the transfer of the problem from 1D to 2D is easy, the adaption of the example is not. Thus a continuation of a fold in a hexagon-branch which has no 1D correspondence has been done. As a consequence the similarities of the command files are negligible and instead of discussing changes we quote the whole file again, see Listing 7. The output is shown in Figure 4a,b.

```
%% 1 - creating a clear working space
close all; keep pphome;
%% 2 - initialising the problem (2D)
p=[];
par=[sqrt(60)*sqrt(3-sqrt(8))+1e-3, 0, 60]; % [lambda, sigma, d]
dom=[4,1]; % domain parameter
p=schnakinit(p,dom,20,par);
p.nc.dsmax=1e-1; % increase dsmax for faster calculation
p.file.smod=1; % store each solution
p=setfn(p,'tr_f2D');
% plot improvments for 2D
```

```
p.plot.pstyle=2;
p.plot.cm=hot;
p.fuha.outfu=@schnakbra_f2D; % new branch data (in particular L8-norm with
    ||1||=1)
kc=sqrt(sqrt(2)-1);
p.Om=16*pi^2*(dom(1)/kc*dom(2)/(sqrt(3)*kc)); % interval length
% names for cmp of schnakbra
p.plot.auxdict={'\lambda','\sigma','d','||u||_{\infty}','min(|u|)','||u||_8'};
%% 3 - find first two bif-points from homog. branch
p.nc.nsteps=30;
p=findbif(p,2); % find first two bif points in max p.nc.nsteps steps, if possible
%% 4 - branch-switch to cold hexagons
p=swibra('tr_f2D','bpt2','hex_f2D',0.05); % switch to cold hexagon branch
p.sw.foldcheck=1; % detect folds
p.sw.bifcheck=0; % disable bif detection
p=cont(p,10); % cont for max of 10 steps
%% 5 - fold continuation
p=spcontini('hex_f2D','fpt1',2,'fold_f2D'); % init fold continuation in par 2
p.sol.ds=-1e-3;  % new stepsize in new primary parameter
p.plot.bpcmp=1;  % plot lam of fold position over sigma in fig 2 now
clf(2);
p.nc.lammin=-10; %p.nc.lammin=-0.5; % set minimal sigma (!) to -0.5
p=cont(p,15); % cont for a max of 15 steps
%% 6 - cont. in lam again from foldpoint
p=spcontexit('fold_f2D','pt9','hex2a_f2D'); % back to normal cont
p.sol.ds=-0.005;
p.nc.dsmax=0.02;
p.nc.lammin=3.2; % minimal lambda (!) is 3.2 now
p.plot.bpcmp=0; % plot l2-norm over lam in fig 2 again
clf(2);
p=cont(p,25);
p=spcontexit('fold_f2D','pt9','hex2b_f2D'); % back to normal cont
p.sol.ds=0.01;
p.nc.dsmax=0.05;
p.nc.lammin=3.208; % set min lambda to approx bif point
p.plot.bpcmp=0; % plot l2-norm over lam in fig 2 again
p=cont(p,10); % cont for a max of 10 steps
%% 7 - plot BD
figure(3);
clf;
% plot analytical trivial branch
plot([3.208,3.24],[3.208,3.24],'color','k','Linewidth',4);
hold on;
plot([3.208,3.05],[3.208,3.05],'color','k','Linewidth',2);
% plot computed branches with l8-norm
plotbra('tr_f2D','pt6',3,6,'cl','k');
plotbra('hex_f2D','pt10',3,6,'fp',2,'cl','b');
plotbra('hex2a_f2D','pt24',3,6,'cl','r');
plotbra('hex2b_f2D','pt9',3,6,'cl','r');
% post processing
axis([3.2 3.242 3.2 3.45]);
text(3.202,3.33,'ch, \sigma=0','color','b','fontsize',16);
text(3.205,3.4,'ch, \sigma=-0.1395','color','r','fontsize',16);%3.205,3.432
text(3.225,3.21,'hom','color','k','fontsize',16);
%% 8 - plot lambda over sigma for fold
figure(4);
clf;
plotbra('fold_f2D','pt15',4,1); % plot lam over sigma for fold position
```

Listing 7: cmds_f2D.m; fold continuation of so called cold hexagons in 2D Schnakenberg model.

(a) plot λ over σ for fold position (b) Fold positions

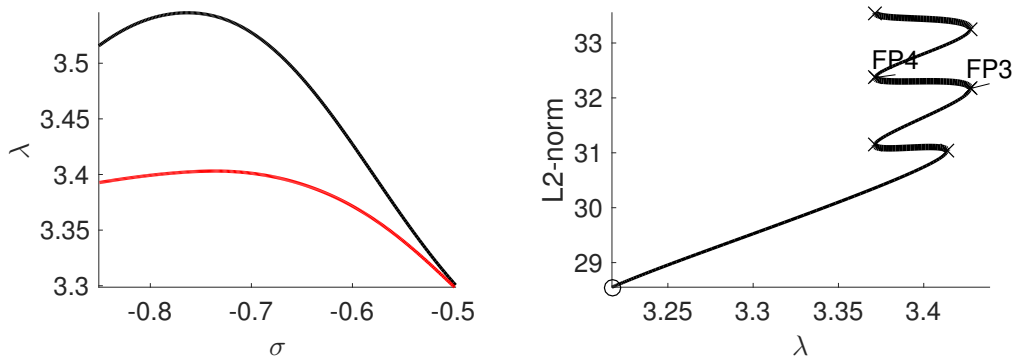


Figure 3: (a) continuation of the fold points FP3 (black line) and FP4 (red line) shown in (b) with respect to σ. (b) third and fourth fold point on the blue branch in Figure 2.

(a) Bifurcation diagram (b) plot λ over σ for the fold position (c) solution at blue fold in a [4,1] domain

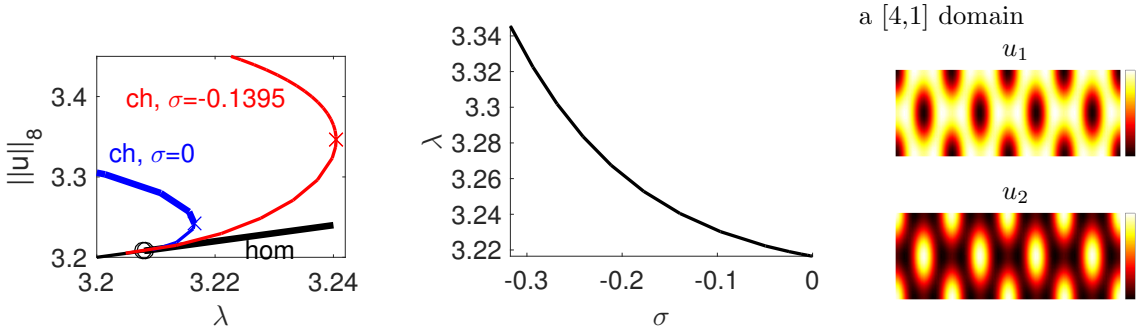


Figure 4: (a),(b) `Matlab` output after the run of `cmds_f2D.m`. ch stands for cold hexagons the form of solution on the blue/red branch. (c) both solution components at the fold on the blue branch.

We used `findbif` instead of `cont` to find the bifurcation points on the trivial branch, as the bifurcation detection methods in `cont` tend to fail here due to too many bifurcations points close to each other. This is a general troubleshoot if one does not find the desired bifurcation point. Changing `p.sol.ds` along with the limits `p.nc.dsmin` and `p.nc.dsmax` or a change of the mesh are two further good tries.

For a better plotting experience we customized `pde2path` to plot the L^8 norm of the first component u_1, normalized through $\|1\|_8 = 1$, instead of the L^2 norm by replacing `stanbra.m` with `snakbra_f2D.m`. In greater detail `snakbra_f2D` adds the L^8 norm to the outputs generated by `stanbra`. This is the convenient way to change the branch data generating function. The first branch data generated by `p.fuha.outfu` should always be the parameters for axis labeling through the use of `auxdict`.

6 Branch point continuation

The idea of fold continuation is to follow a zero eigenvalue via the extended system (12) in [RU17]. Thus we may attempt to use the same method for branch point continuation. Although this is only guaranteed to work for systems (bifurcations) with up-down symmetry, see [WS84], in practice we found it to work for general systems with some caveats.

The syntax keeps the same, e.g. call `spcontini` at the branch point which should be continued, continue with `cont` and call `spcontexit` to leave the branch point continuation and be able to start with a normal continuation again. As an example we continued the branch point of a periodic

solution from the trivial one in 1D, marked as BP1 in Figure 2a. See Figure 5 for the results. The

(a) Bifurcation diagram (b) plot d over λ for branch point position

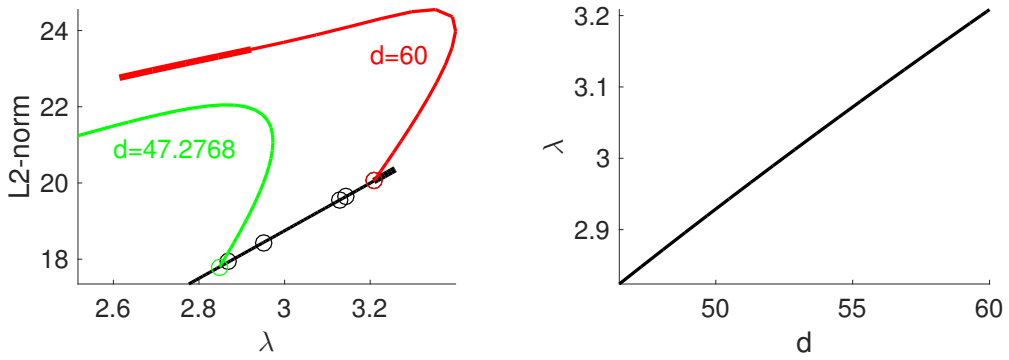


Figure 5: (a) Continuation of the branch point connecting the trivial and an periodic solution. Shown is the trivial branch and the periodic branch for $d = 60$ and $d = 47.2768$. (b) The position of the branch point with respect to d. Both plots are generated though `cmds_b1D.m`.

code is given in `cmds_b1D.m`. As the Jacobian in `spjac` is ill conditioned for this branch point a secant predictor is used through the switch `p.sw.secpred=1`. `bpcontexit` is used to mark the point as a branch point and to compute the pertinent tangent.

7 Details of `plotbra`

All relevant data for branch plotting is saved in a matrix in `p.branch`. Each column represents a computed – not necessarily stored – point while the rows contain different informations about this point. The first five rows are filled by `bradat` and are the point number, its type (reg/bif/etc), the number of negative eigenvalues, the active parameter value, the solutions error and its L^2-norm. The further rows are by default filled through `stanbra` which can easily be changed through `p.fuha.outfu` as for example done in §5 with the function `schnakbra_f1D`.

Thus this data can be used to directly plot branches with the `Matlab` plotting commands. To simplify this `pde2path` has the high level function `plotbra` which plots and in particular labels branches suitable for most situations.

The basic call of `plotbra` is `plotbra(p)` for a problem structure or `plotbra('dir')` to plot the branch from a directory. See Figure 6a for an example. The generating code is given in Listing 8, cell 'basic 2'. The behavior of `plotbra(p)` respectively `plotbra('dir')` can be controlled via a few fields in `p.plot`, see Table 2. This fields should be set in the beginning of a continuation for example in the init file.

Table 2: Options of `plotbra` which can be set through structure fields.

option	field	purpose	default
lsw	p.plot.lsw	switch for basic labeling, see below and [dWDR+17],tab 24 for details; lsw= $xxxx1_2$ is required for user lambda labels	1
wnr	p.plot.brafig	figure number to plot	3
cmp	p.plot.bpcmp	row of p.branch which shall be plot. If number: y-comp only, if vector: [x y] comp number	0
fancy	p.plot.fancybd	fancyness of plot, in particular: 0: no annotation arrow; 1: fixed ones; 2: movable ones	1
fs, lfs	p.plot.fs	axis and label font size	16
auxdict	p.plot.auxdict	dictionary for the components in `stanbra`, see below for details	{}

The option `lsw` is the main switch for default plotting. Written in binary number its entries are

switches for labels of regular, fold, hopf, branch and usrlam points, i.e. a value of $10000_2 = 16_{10}$ correspond to labeling all regular points, while $01100_2 = 12_{10}$ correspond to labels for all fold and hopf points only. The value has to be given in decimal numbers. See Table 3 for a list of possible values.

Table 3: Settings for `p.plot.lsw` and `'lsw',lsw` argument of `plotbra`, for regular point labels='off'. For regular point labels='on', add 16 to lsw.

lsw	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
userlam	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
branch	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
hopf	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
fold	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

The default value for `lsw` labels so called 'user lambdas' only, which have to be defined in the beginning of a continuation as a vector in `p.usrlam`. See Figure 6c and cells usrlam 1,usrlam 2 in Listing 8 for a minimal example. Further modifications of the branch plotting can be achieved through (string,value) pairs as additional inputs, see Table 4 for a description of these.

Table 4: (string, value) pairs for function `plotbra`.

option	purpose	standard value
lwun	line width of unstable solution	2
lwst	line width of stable solution	4
lw	overrides lwst=lw, lwun=lw/2	set through lwun, lwst
tyun	line type of unstable solution	'-'
tyst	line type of stable solution	'-'
ms	marker size (branch/hopf points)	5
fms	marker size (fold points)	0
lms	marker size (labeled points); if number: all labeled points, if vector: [reg. bif.]	[5 4]
rms	marker size (all reg. points) marks reg points; 0 is the convenient choice	0
lsw	recessive switch for the basic labeling, detailed description above; lsw= $xxxx1$ is required for user lambda labels	p.plot.lsw=1
lab	place markers and labels for given label-list and disables the labels for user lambdas	set through lsw
labi	as 'lab', but step through all labels in branch with given increment	
labu	if 1 & lsw= $1 + x$: user lambdas will be plotted even if a set of regular labels is given through 'lab' or 'labi'	0
fs	font size for labels and axes	p.plot.fs
lfs	font size for labels, if zero no labels (only markers)	p.plot.fs
fp	first point to be considered	1
lp	last point to be considered	last point in p.branch
cl	color, see option in plot	'black'
bplab	place markers and labels for label-list of branch points	set through lsw
fplab	place markers and labels for label-list of fold points	set through lsw
hplab	place markers and labels for label-list of hopf points	set through lsw
fancy	0: no annotation arrow, 1: static annotation arrow, 2: annotation arrow movable by mouse	1
wnr	figure number	p.plot.brafig=3
cmp	row of p.branch which shall be plot. If number: y-comp only, if vector: [x y] comp number	p.plot.bpcmp=0 ($\|u_1\|_2$)

os	length of annotation arrow	1
odr	direction of annotation arrow for regular points fully random if [0 0]	[1 -1]
ods	as odr for bifurcation points	[1 0.1]
auxdict	dictionary for the components in `stanbra`, see below for details	{}

They are called as `plotbra(...,'option',value)`, like `plotbra(p,'fancy',2,'lab',[4 5])`. Additional one can also insert an arbitrary amount of (string,value) pairs as a cell array, which can be used to easily recall personal standard settings, for instance if other marker/branch sizes are desired. See Figure 6b and Listing 8 cell 'cell 2' for an example. As commonly used and for backward compatibility one can also call `plotbra(X,wnr,cmp,varargin)` with `X=p` or `'dir'` or `'dir,'pt'` to set the figure and component number without the call of the option string. If no point is selected the last point in the directory will be loaded. If the directory is not messed up the specification of a point can always be replaced through the last point option, 'lp', but if one wants to plot the branch until a special point, for instance a branch point, the specification as a point in the directory is more easy, as one can use the syntax `plotbra('dir','bpt1')`.

Only the last call of an option is taken into account, thus (string,value) pairs always overrides settings through fields in `p.plot` and of course default settings as well. The only exception are the first four digits in 'lsw' written as a binary number, as this options are dominated by any other labeling option. Still only the last call of 'lsw' is taken into account and lsw= $xxxx1_2$ is necessary to label user lambdas.

The option 'cmp', respectively `p.plot.bpcmp` allows negative numbers. In greater detail 0 stands for the last component in `bradat`, e.g. the L^2-norm, while -5 is the first component, e.g. the points number. The positive values for 'cmp' count through the user datas generated by `p.plot.outfu`.

The option 'auxdict' respectively the field `p.plot.auxdict` can be filled with a dictionary of the components generated by `p.fuha.outfu`, as already used in the `schnakinit` file, see Listing 4 line 12. If set it is used to label the x-axis in the default setting as well. For this reason the dictionary should start with the parameters, i.e. {'\lambda', '\sigma',d, ...}, which implies, that `p.fuha.outfu` should generate this data as the first components as well.

The option 'fancy' respectively `p.plot.fancybd` switches the style of annotations. The setting ('fancy',0) does not plot any annotation arrows. The default setting 'fancy'=1 plots fixed annotation arrows and is the convenient choice when working with `pde2path`. When preparing papers one can spare a lot post processing time with the use of 'fancy'=2 which will plot movable annotation, but this is slow for many labels. Also 'fancy'=2 uses undocumented `Matlab` code and thus is error prone. In particular 'fancy'=2 does not work satisfactorily with subplots.

```
%% basic/cell 1 - switch to periodic branch
p=swibra('tr_plot','bpt1','per_plot',1e-2); % switch to new branch with ds=0.01
p=cont(p,150); % continuation for a maximum of 150 steps
%% basic 2 - basic plot
figure(3);
clf;
plotbra('tr_plot'); % trivial branch
% plotbra('tr_plot','pt20'); % plots the same, 'pt10' plots a shortend branch
plotbra('per_plot'); % periodic branch
% plotbra(p); % generates the same as plotbra('per_plot')
%% cell 2 - plot with options through a cell array
figure(3);
clf;
basic={'ms',2,'fancy',0,'lsw',15};
plotbra('tr_plot',basic); % trivial branch
% plotbra('tr_plot','ms',2,'fancy',0,'lsw',15); % plots the same
plotbra('per_plot',basic); % periodic branch
```

```
% plotbra(p,basic) % plots the same as plotbra('per_plot',basic)
%% usrlam 1 - usrlam data generation
p=swibra('tr_plot','bpt1','per_plot_usrlam',1e-2); % switch to periodic branch
p.usrlam=3:0.1:3.5; % set userlambdas
p=cont(p,200); % continuation for a maximum of 200 steps
%% usrlam 2 - usrlam plotting
figure(3);
clf;
plotbra(p); % periodic branch with usrlabels, as lsw=1 by default
% plotbra('per_plot'); would plot the same
plotbra('tr_plot'); % trivial branch - no labels, as p.usrlam={} in Cell 3
```

Listing 8: `cmds_plot.m`; differnt plots of the trivial and the first periodic branch generated in `cmds_f1D`, see Listing 6. The data generation in the first 12 lines is the same as in `cmds_f1D` besides the problem directory changed to `tr_plot`. The cell headings descripe which cell is necessary for the corresponding examples in Figure 6. Lines 1-12 re required for all examples.

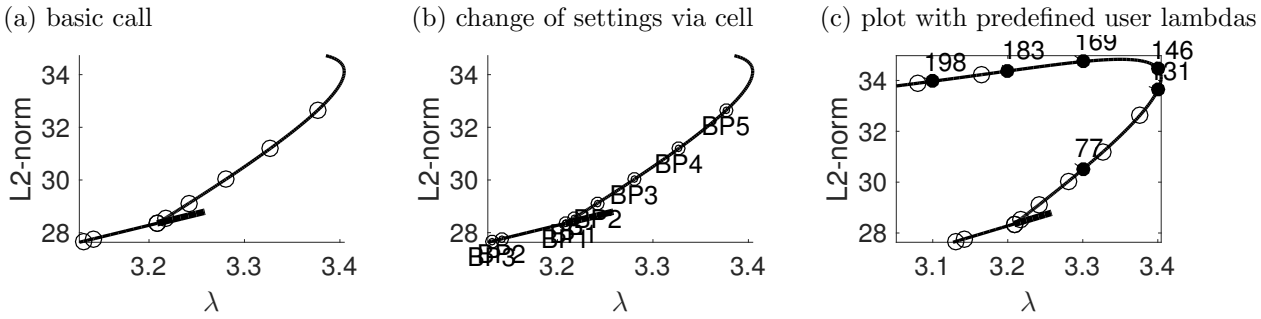


Figure 6: (a) basic call of `plotbra(p)`. (b) call of `plotbra` with reduced markersize and no annotation arrows via an cell array of (string,value) pairs. (c) call of `plotbra` with predefined user lambdas 3,3.1,3.2,3.3,3.4 and 3.5. For all three examples see `cmds_plot.m`.

References

[DRUW14] T. Dohnal, J. Rademacher, H. Uecker, and D. Wetzel. pde2path - V2: faster FEM and periodic domains, 2014.

[dWDR+17] H. de Witt, T. Dohnal, J. Rademacher, H. Uecker, and D. Wetzel. pde2path - Quickstart guide and reference card, 2017.

[Prü16] U. Prüfert. OOPDE: FEM for Matlab, `www.mathe.tu-freiberg.de/nmo/mitarbeiter/uwe-pruefert/software`, 2016.

[RU17] J. Rademacher and H. Uecker. The OOPDE setting of pde2path - a tutorial via some Allen-Cahn models, 2017.

[Uec17] H. Uecker. pde2path, `www.staff.uni-oldenburg.de/hannes.uecker/pde2path`, 2017.

[UW14] H. Uecker and D. Wetzel. Numerical results for snaking of patterns over patterns in some 2D Selkov-Schnakenberg Reaction-Diffusion systems. *SIADS*, 13-1:94–128, 2014.

[UWR14] H. Uecker, D. Wetzel, and J. Rademacher. pde2path – a Matlab package for continuation and bifurcation in 2D elliptic systems. *NMTMA*, 7:58–106, 2014.

[Wet17] D. Wetzel. A pde2path `plotsol` tutorial, 2017.

[WS84] B. Werner and A. Spence. The computation of symmetry-breaking bifurcation points. *SIAM J. Numer. Anal.*, 21(2):388–399, 1984.