pde2path – plotsol tutorial

Daniel Wetzel[1]

[1] Universität Oldenburg

March 27, 2017

Abstract

The function `plotsol` is used in `pde2path`'s main function `cont` to plot the current solution, and the user can use `plotsol` to plot results a posteriori. Here we describe some details of `plotsol`'s functionality.

1 Preparations

First we need to prepare some solutions to describe `plotsol`'s functionalities, in 1D, 2D, and 3D, such that we work in the `OOPDE` setting, see [RU17]. In `PlotsolDemo`, which can be found in `pde2path`'s demo folder, we consider the Schnakenberg model

$$\partial_t u_1 = \Delta u_1 - u_1 + u_1^2 u_2,$$
$$\partial_t u_2 = d\Delta u_2 + \lambda - u_1^2 u_2,$$

where $d = 60$, and λ is used as bifurcation parameter, see also [dW17].

`PlotsolDemo`'s first script `cmds.m` is shown in Listing 1. Calling `schnakinit(p,dim)` sets $\lambda = 3.3$ and $(u, v) = (\lambda, 1/\lambda)$ so that we start on a homogeneous branch. From the theory we know that this staring point is near a Turing bifurcation. Here `dim` is the space dimension. We run `findbif` to find the Turing bifurcation, then `swibra` to switch to non-homogeneous solutions, and `cont` to follow the branches.

```
close all; keep pphome;
%%
% creating 1D files
p1=[];p1=schnakinit(p1,1);p1.nc.nsteps=10000;p1=findbif(p1,1);
p1=swibra('h1','bpt1','s1');p1=cont(p1,20);
% creating 2D files
p2=[];p2=schnakinit(p2,2);p2.nc.nsteps=10000;p2=findbif(p2,2);
p2=swibra('h2','bpt2','s2');p2=cont(p2,3);
% creating 3D files
p3=[];p3=schnakinit(p3,3);p3.nc.nsteps=10000;p3=findbif(p3,1);
p3=swibra('h3','bpt1','s3');p3=cont(p3,2);
```

Listing 1: Command script `cmds.m` for preparing some 1D, 2D, and 3D solutions

2 Plotting

Let `p` be `p1`, `p2`, or `p3` created in `cmds.m`. The solution is stored in `p.u`. In older versions of `pde2path` it was necessary to call `plotsol` in the form `plotsol(p,wnr,cnr,pst)`, where `wnr`, `cnr`, and `pst` are for setting the figure number, component number, and plotting style, respectively. Now it is enough to call `plotsol(p)`. If the user is doing this, then `plotsol` checks, if `p.plot` has the structure fields which are listed in Table 1. If this is the case, then `plotsol` uses this information and if not, then it uses default settings.

We have three possibilities to change for instance the option `pstyle` from 1 to 2. These are

Table 1: `plotsol`'s list of general options

option name	meaning	possible entries	default setting
`pfig`	figure nr.	integer	1
`pcmp`	component nr.	integer	1
`pstyle`	plotting style	integer or string	1
`fs`	fontsize	integer	16

- `plotsol(p,11,1,2)`
- `p.plot.pstyle=2; plotsol(p)`
- `plotsol(p,'pstyle',2)`

We can change the other options in a similar way.

Let `s` be the folder `s1`, `s2`, or `s3`. It was necessary to call `plotsolf('s','pt10',11,1,1)` to plot the first component of the 10th solution of the folder `s` into the 11th figure. Now calling `plotsol('s','pt10')`, `plotsol('s1','pt10',11)`, `plotsol('s','pt10',11,1)`, and `plotsol('s','pt10',11,1,1)` gives the same result for this example. The function `plotsolf` is obsolete now, but it still exists and can be used for running scripts of older `pde2path` versions.

`plotsol` plots the last point with highest label if the first argument is a folder and the second not a specific point in this folder (a call like `plotsol('s')` or `plotol('s','lw',3)`). The point `pt1` is not saved in `s`, since we set the switch `smod`=10. If we call something like `plotsol('s1','pt1')`, then the software informs us that this point is not saved in `s`, shows which points exist in `s`, and we can choose one of these points.

Remark 2.1. If one considers a special problem for which `plotsol` cannot be used directly, then one has to write special plotting routines. For this case it is useful to understand `plotsol`'s code. The old version of `plotsol` is short and it is easy to find the main plotting routines implemented there. `plotsol`'s new code became more complex by adding all new options and functionalities so that it is not easy to understand for a new user. For such problems we keep the old simple `plotsol` and `plotsolf` versions in the `lib` and call them `splotsol` and `splotsolf`, respectively.

2.1 1D

For 1D plots `plotsol` uses matlab's function plot, which draws a line. This line is solid, dashed, and dotted if we set `pstyle` to 1, 2, and 3, respectively. If we set `pstyle` via `p.plot.pstyle` or by using the string `'pstyle'`, then we can also use `'-'`, `'--'`, and `':'` instead of 1, 2, and 3, respectively.

There are two extra options for 1D plots, which are listed in Table 2. The first is to change the line width and the second is to change the line color. These options can be controlled via structure field entries in `p.plot` or string arguments in `plotsol`.

Table 2: 1D extra options

option name	possible entries	meaning	default settings
`lw`	integer	line width	1
`cl`	vector or string	line color	[0 0 0]

The color can be chosen via RGB code vectors or the standard matlab strings like for instance 'r' or 'red' for red. Furthermore, `pde2path` has its own predefined color shades, which are listed in Table 3.

Table 3: Color map

color shade	string
green	'g1', 'g2', 'g3'
blue	'b1', 'b2', 'b3'
yellow	'y1', 'y2', 'y3'
red	'r1', 'r2', 'r3'
violet	'v1', 'v2', 'v3'
brown	'br1', 'br2', 'br3'
orange	'o1', 'o2', 'o3'
gray	'gr1', 'gr2', 'gr3'

We can choose the component number which we want to plot via the option `pcmp` as already mentioned above. This means `plotsol` plots the component u_n if we set `pcmp` to n. Furthermore, `plotsol` plots $u_{m_1}, \dots, u_{m_k}$ together into one figure if we set `pcmp` to $[m_1, \dots, m_k]$. This works only for 1D plots. The argument corresponding to `cl` must be a cell array or a matrix (see l14 and l17 of Listing 2). Examples can be found in the file `plot1d.m`, which in the last cell also gives an example how to 'low-level' plot directly without using `plotsol`.

```
   %% Plot solution which is stored in p1. Here we do not change any
   %  plotting option. We did not create the structure fields
   %  lw and cl in p1.plot so that plotsol uses the default settings.
   %  We have already created p1.plot.pcmp=1 and p.plot.pstyle=1
 5 %  and plotsol uses this information.
   plotsol(p1)
   %% Plot the same as in the cell before into figure 12. Use the orange
   %  color shade o1 and the line width 10.
   plotsol(p1,12,'cl','o1','lw',10)
10 %% Plot both components of the solution type pt with highest label, which
   %  can be found s1, into figure 13. Use the plotting styles 2 (dashed) and
   %  1 (solid) and the green colors shades g1 and g2 for u1 and u2,
   %  respectively. Use line width 4 for both components and font size 30.
   plotsol('s1',13,[1 2],[2,1],'cl',{'g1','g2'},'lw',4,'fs',30)
15 %% Plot both components of s1/pt10 into figure 14. Use line widths 1
   %  and 4 and line color red and blue for u1 and u2, respectively.
   map=[1 0 0;0 0 1];plotsol('s1','pt10',14,[1 2],'cl',map,'lw',[1 4]);
   title('');legend('u_1','u_2')
   %% Here we plot the same as in the cell before without using plotsol.
20 figure(15);clf;p=loadp('s1','pt10');
   u1=p.u(1:p.np);u2=p.u(p.np+1:2*p1.np);[po,~,~]=getpte(p);
   plot(po,u1,'color','r','linewidth',1);hold on;
   plot(po,u2,'color','b','linewidth',4);
   legend('u_1','u_2');set(gca,'fontsize',16);axis tight;
```

Listing 2: Command script `plot1d.m` for plotting some 1D solutions

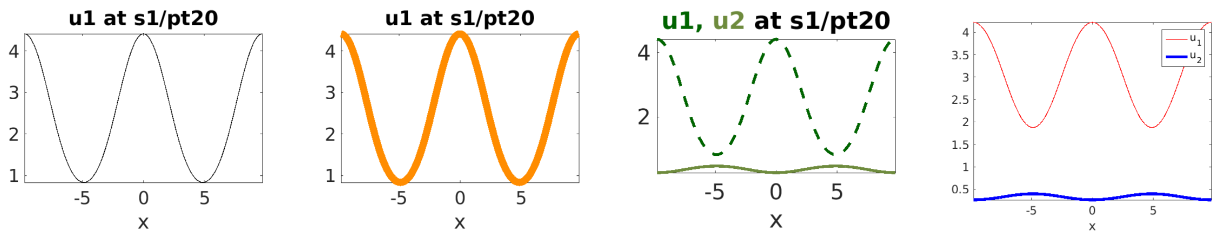

Figure 1: Plots, which are created by running `plot1d.m`

2.2 2D

There are two extra options in `plotsol` for 2D plots, listed in Table 4, namely the color map and the axis. Which predefined color maps exist, how to create own color maps, and which axis styles exist can be found in matlab's help function.

Table 4: 1D extra options

option name	possible entries	meaning	default settings
`cm`	string or matrix	color map	cool
`axis`	string	axis style	'tight'

`plotsol` plots the used mesh, 3D mesh, 2D contour, and 3D surface plot if we set `pstyle` to 0, 1, 2, and 3, respectively, see Listing 3 and Fig. 2.

```
%% We plot always u1 in the following.
%  We set p2.plot.axis='image' in schnakinit.m.
plotsol(p2,'pstyle',0); % Use default setting and plotting style 0
plotsol(p2,12,'pstyle',1); % Use figure 12 and plotting style 1
% Use figure 13, plotting style 2, and color map hot.
% Do not show any labels for the x- and y-axis.
plotsol(p2,13,'pstyle',2,'cm',hot);xlabel('');ylabel('');
% Use figure 14, plotting style 3, and axis style 'normal'.
plotsol(p2,14,'pstyle',3,'axis','normal');
```

Listing 3: Command script `plot2d.m` for plotting some 2D solutions

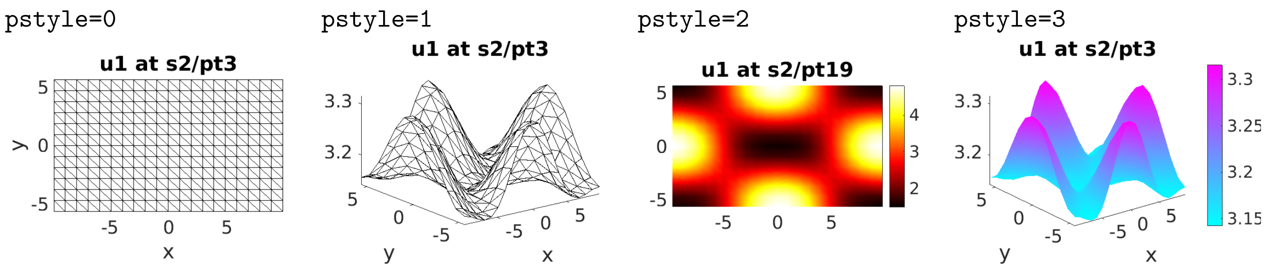

Figure 2: Different plotting styles

2.3 3D

For solutions over 3D domains `plotsol` plots a slice, isosurface, and face plot for `pstyle`= 1, 2, and 3, respectively. We can control the color map for face and slice plots via the option `cm` like in 2D. There are no further options for these two plotting styles. `plot3d.m` (see Listing 4) shows how to plot some solutions. The resulting plots are shown in Figure 3.

Examples for creating isosurface plots are also shown in `plot3d.m`. The resulting plots are shown in Figure 4. For this style we have some options which can be found in Table 5. The user can give level values via the option `lev`. Another way to control the levels works via the option `levn`. If we set for instance `pcmp`=j and `levn`=k, where k is a positive integer, then `plotsol` plots k levels of u_j. The m-th level l_m is given by

$$l_m = \min(u_j) + \frac{\max(u_j) - \min(u_j)}{k+1}\, m.$$

Fine controlling of one or two levels can be done as follows. If $0 < k < 1$, then `plotsol` plots two levels, which are given by

$$\begin{aligned} l_1 &= \min(u_j) + (\max(u_j) - \min(u_j))k, \\ l_2 &= \min(u_j) + (\max(u_j) - \min(u_j))(1-k). \end{aligned}$$

If $-1 \le k < 0$, then `plotsol` plots one level, which is given by

$$l_1 = \min(u_j) + (\max(u_j) - \min(u_j))(-k).$$

If one of the structure fields `lev`, `levn`, or `levc` in `p.plot` exist and the user wants to remove this field, then one can do this by setting it to 0. This also works via matlab's function `rmfield`. If one sets `levn` via the string argument, then `plotsol` ignores the information which is in `p.plot.lev` and vice versa. If `lev` and `levn` are both given via the structure fields in `p.plot`, then `plotsol` uses `lev` and ignores `levn`. A conflict occurs if `lev` and `levn` are both passed to `plotsol` via string arguments. In this case `plotsol` ignores `lev` and uses `levn`$= 1/4$.

The colors of the levels can be set via the option `levc`. This can be done in two ways. If we plot m levels and in `levc` are stored more than two colors, then `plotsol` uses the first m stored colors. If only two colors are stored in `levc`, then `plotsol` does a color movement from the first color to the second. This means that if for instance `levn`$= 3$ and blue and red are stored in `levc` like in the default case (if `levc`$= 0$ or does not exist), then the color of the first, second, and third level are blue, violet, and red, respectively.

Table 5: 3D extra options if `pstyle`$= 1$

option name	possible entries	meaning	default setting
`alpha`	$0 \le \alpha \le 1$	transparency	1
`ng`	integer	number of discretization points per space direction for computing isosurfaces	20
`lev`	vector of real numbers	levels for the isosurfaces	0
`levn`	$k \in \mathbb{Z}_+$ or $-1 \le k < 1$	$k \ge 1$: number of levels $0 \le k < 1$: two levels $-1 \le k < 0$: one level	1/4
`levc`	matrix or cell	level colors	[0 0 1; 1 0 0]

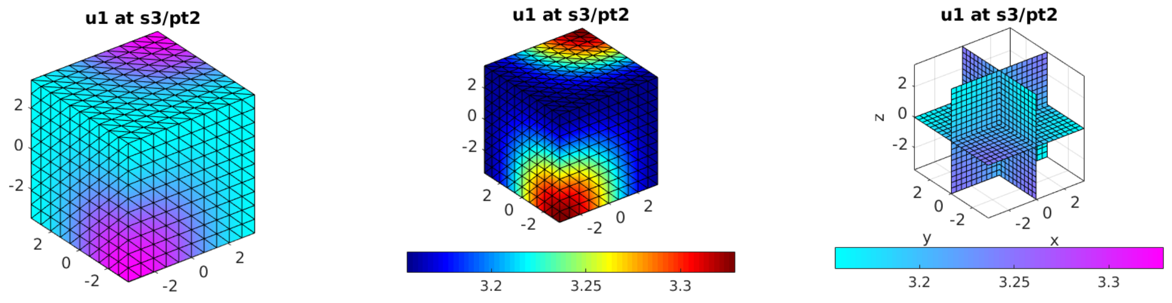

Figure 3: 3D face and slice plots, which are created in first three cells of `plot3d.m`

```
%% pstyle=1 und 3
%  Plot the first component into figure 7 and use the plotting style 3.
%  Do not show the color bar.
%  We plot always the first component in the following.
plotsol(p3,7,1,3);colorbar('off');
%% Use figure 8, plotting style 3, and color map 'jet'.
plotsol(p3,8,1,3,'cm','jet');
%% Use figure 9 and plotting style 1 (here we use default settings).
plotsol(p3,9);
%% pstyle=2
%  Use figure 11 and plotting style 2.
```

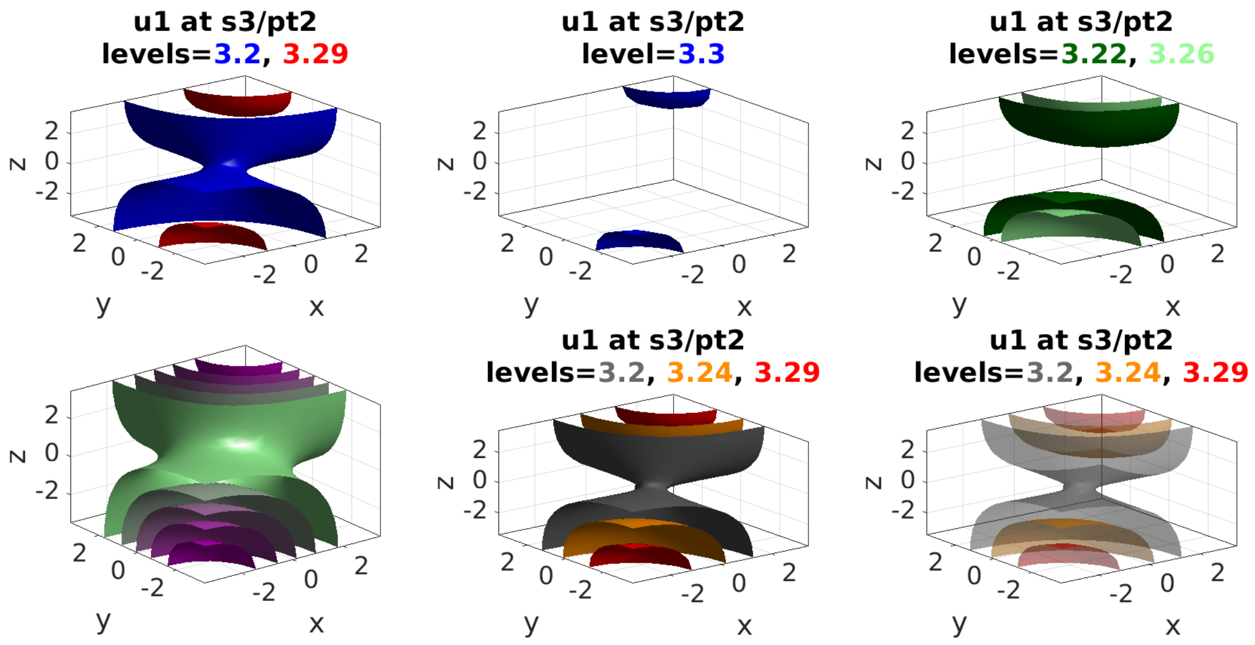

Figure 4: 3D isosurface plots, which are created in the cells 4-9 of `plot3d.m`

```
%  We use always plotting style 2 in the following.
plotsol(p3,11,1,2);
%% Use figure 12 for plotting only one level, which is given by
%  l=min(u1)+(max(u1)-min(u1))*k with k=5/6.
plotsol(p3,12,1,2,'levn',-5/6);
%% Use figure 13 for plotting the levels l1 and l2, which are given by
%  l1=min(u1)+(max(u1)-min(u1))k,
%  l2=min(u1)+(max(u1)-min(u1))(1-k)
%  with k=0.4, respectively.
%  Use dark and light green shades for l1 and l2, respectively.
p3.plot.levc={'g1' 'g3'};plotsol(p3,13,1,2,'levn',0.4);
%% Use figure 14 for plotting five levels, which are given by
%  lm=min(u1)+m*(max(u1)-min(u1)/(k+1) with k=5.
%  Do a color movement from the green shade g3 to the violet shade v2.
%  Do not show any figure title.
plotsol(p3,14,1,2,'levn',5,'levc',{'g3','v2'});title('');
%% Plot the isosurface levels 3.2 (gray), 3.24 (orange), and 3.29 (red)
%  into figure 15.
plotsol(p3,15,1,2,'lev',[3.2 3.24 3.29],'levc',{'gr1' 'o1' 'r'});
%% Use figure 16 for doing the same as in the cell before
%  and set the intensity to 0.5.
plotsol(p3,16,1,2,'lev',[3.2 3.24 3.29],'levc',{'gr1' 'o1' 'r'},'alpha',0.5);
```

Listing 4: Command script `plot3d.m` for plotting some 3D solutions

References

[dW17] H. de Witt. Fold and branch point continuation in pde2path – a tutorial for systems, 2017.

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