Periodic boundary conditions in `pde2path`

Tomáš Dohnal[1], Hannes Uecker[2]

[1] Institut für Mathematik, MLU Halle–Wittenberg, D06099 Halle (Saale), tomas.dohnal@mathematik.uni-halle.de
[2] Institut für Mathematik, Universität Oldenburg, D26111 Oldenburg, hannes.uecker@uni-oldenburg.de

April 30, 2018

Abstract

We describe the implementation of periodic boundary conditions in `pde2path` 2.3, and give examples on their use in some scalar model problems in 1D, 2D and 3D.

Contents

1 Introduction

The tutorial [RU18] gives an elementary introduction to the new `OOPDE` setting of `pde2path` [UWR14, Uec18], using the (cubic–quintic) steady Allen–Cahn equation

$$G(u) := -c\Delta u - \lambda u - u^3 + \gamma u^5 \stackrel{!}{=} 0, \tag{1}$$

and variants of this as model problems, where $u = u(x) \in \mathbb{R}$, $x \in \Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, Ω an interval, a rectangle or cuboid, respectively, and where we consider various boundary conditions (BC). Here we use some variants of (1) to explain the use of *periodic BC* (pBC) in a simple scalar setting, again in 1D, 2D and 3D. At many places we refer to [RU18] for background, and thus recommend to new users to start with the demos explained in [RU18]. Some other `pde2path` demos also use periodic BC, namely `nlb, schnaktravel`, and `twoFluid`, and problems with pBC have for instance been treated with `pde2path` in [ZHKR15, DU16]. The software, including a number of demo directories, documentation, tutorials and a quickstart guide [dWDR$^+$18] with installation instructions and a demo overview can be downloaded from [Uec18]. The demo directories for this tutorial are in `demos/acpbc`.

In §2 we briefly review the algorithm used to implement periodic BC by transforming the finite element matrices (stiffness and mass matrix) as well as the load vector from homogeneous Neumann BC to periodic BC. This discussion follows to some extent that in §2.6 of [DRUW14] but extends it to all dimensions 1, 2, and 3. In §3 we then give the tutorials on how to use these pBC in 1D, 2D and 3D for variants of (1).

2 Transform from homogeneous Neumann BC to periodic BC

2.1 Mathematical algorithm

We start with the one dimensional case with $\Omega = (a, b) \subset \mathbb{R}$. With Neumann BC the values of the finite element solution `u` at $x = a$ and $x = b$ are unknowns to be solved for. The finite element basis functions (hat functions) ϕ_1 and ϕ_{n_p} corresponding to the nodes $x_1 := a$ and $x_{n_p} := b$, respectively, are different from those at the interior points $x_2, \ldots, x_{n_p-1}$. Namely, they are "incomplete" as they lack contributions from $[a-h, a)$ and $(b, b+h]$ respectively, see Fig. 1 (a). In the case of periodic BC,

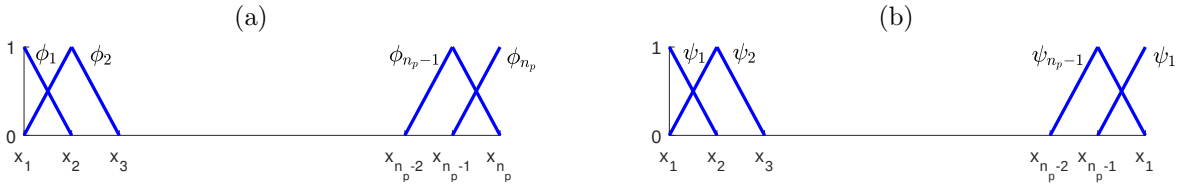

Figure 1: Finite element hat functions in one dimension on an interval with Neumann BC in (a) and with periodic BC in (b).

where $x = a$ and $x = b$ are identified (and the node x_{n_p} dropped from the grid), the basis function at $x = a$ is given by ϕ_1 and ϕ_{n_p}. Denoting the hat functions in the periodic setting by ψ_j, we have

$$\begin{aligned}\psi_j &= \phi_j, \ j = 2, \ldots, n_p - 1,\\ \psi_1 &= \phi_1 + \phi_{n_p}|_{[a,b)}.\end{aligned}$$

The finite element matrices for the periodic case can thus be generated from those for the homogeneous Neumann case by adding up corresponding entries, e.g., for the mass matrices M^{per} and M^{Neum}

$$\begin{aligned}
M^{\text{per}}_{ij} &= (\psi_i, \psi_j)_{L^2(\Omega)} = (\phi_i, \phi_j)_{L^2(\Omega)} = M^{\text{Neum}}_{ij}, \quad i, j \in \{2, \ldots, n_p - 1\},\\
M^{\text{per}}_{1j} &= (\psi_1, \psi_j)_{L^2(\Omega)} = (\phi_1 + \phi_{n_p}, \phi_j)_{L^2(\Omega)} = M^{\text{Neum}}_{1j} + M^{\text{Neum}}_{n_p j}, \quad j \in \{2, \ldots, n_p - 1\},\\
M^{\text{per}}_{j1} &= (\psi_j, \psi_1)_{L^2(\Omega)} = M^{\text{Neum}}_{j1} + M^{\text{Neum}}_{jn_p}, \quad j \in \{2, \ldots, n_p - 1\},\\
M^{\text{per}}_{11} &= (\psi_1, \psi_1)_{L^2(\Omega)} = (\phi_1 + \phi_{n_p}, \phi_1 + \phi_{n_p})_{L^2(\Omega)} = M^{\text{Neum}}_{11} + M^{\text{Neum}}_{1n_p} + M^{\text{Neum}}_{n_p 1} + M^{\text{Neum}}_{n_p n_p}.
\end{aligned}$$

For the stiffness matrix the modification is completely analogous. These modifications are in `pde2path` carried out efficiently by a matrix multiplication. Let us, for illustration, take the case $n_p = 4$ with $x_1 = a, x_4 = b$. The transformation is then carried out via

$$\begin{aligned}\texttt{M}^{\text{per}} &= \texttt{p.mat.fill}' * \texttt{M}^{\text{Neum}} * \texttt{p.mat.fill},\\ \texttt{K}^{\text{per}} &= \texttt{p.mat.fill}' * \texttt{K}^{\text{Neum}} * \texttt{p.mat.fill},\end{aligned} \tag{2}$$

where

$$\texttt{p.mat.fill} = \begin{pmatrix} 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 0 & 0 \end{pmatrix}.$$

The multiplication by $\texttt{p.mat.fill}'$ from the left adds up the pertinent rows (here rows 1 and 4) and drops the redundant row 4. The right multiplication by `p.mat.fill` does the same with the columns. For the load vector $F^{\text{per}} = ((f, \psi_1)_{L^2(\Omega)}, \ldots, (f, \psi_{n_p-1})_{L^2(\Omega)})^T$ we have analogously

$$\texttt{F}^{\text{per}} = \texttt{p.mat.fill}' * \texttt{F}^{\text{Neum}}.$$

To recover the solution vector `u` on the full grid $(x_1, \ldots, x_{n_p})$, e.g. for plotting purposes, one performs simply $\texttt{p.mat.fill} * \texttt{u}^{\text{per}}$. `pde2path` also provides the matrix `p.mat.drop` for dropping the

redundant components of a vector (e.g. a vector of grid points on a Neumann domain). For the above example

$$\texttt{p.mat.drop} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The above example of `p.mat.fill` and `p.mat.drop` applies only in the case of one equation (`p.nc.neq`=1 in `pde2path`). For a system of equations the above matrices are simply repeated as blocks on the diagonal of the full matrix.

In higher dimensions `pde2path` assumes that the boundary segments to be identified via the periodic BC are flat and aligned with the coordinate axes (typically the domain Ω is a rectangle in 2D and a cuboid in 3D but, e.g., if in 2D the periodic BC are to be imposed only in the x-direction, then only the part of $\partial\Omega$ with the minimal and the maximal values of x must consist of two parallel lines spanning the same y-interval, see Fig. 2 for an example). With periodic BC in the x_j-direction

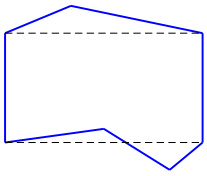

Figure 2: Example of an admissible domain in 2D with periodic BC only in the x-direction. $\partial\Omega$ is plotted by the full blue line.

($j \in \{1,2,3\}$) the x_i-coordinates with $i \neq j$ of the boundary points on the x_j-sides have to be identical in order to match the boundary point. The procedure is analogous to 1D: the finite element basis functions at the retained boundary segments consist of the sum of the "incomplete" Neumann basis functions on the opposite boundary sides.

In the case of periodic BC in more than one direction the transformation matrices `p.mat.fill` and `p.mat.drop` can be written as the product of matrices for each direction. For instance, for periodic BC in x and y

$$\texttt{p.mat.fill} = \texttt{A}_\texttt{x} * \texttt{A}_\texttt{y}, \quad \texttt{p.mat.drop} = \texttt{B}_\texttt{y} * \texttt{B}_\texttt{x}, \tag{3}$$

where $\texttt{A}_\texttt{x}$and $\texttt{B}_\texttt{x}$ are the `fill` and `drop` matrices, respectively, for periodic BC in the $x-$direction, and $\texttt{A}_\texttt{y}, \texttt{B}_\texttt{y}$ are for the y-direction.

2.2 Internal `pde2path` commands for using periodic BC

The periodic BC to be used are encoded in the switch `p.sw.bcper`. The possible values are

`p.sw.bcper` = `j`, periodic BC in the $j-$direction, $j \in \{1,\ldots,d\}$ in d-D ($d \leq 3$)

`p.sw.bcper` = [`i j`], periodic BC in the directions i and j; $i,j \in \{1,\ldots,d\}$ in d-D ($2 \leq d \leq 3$)

`p.sw.bcper` = [1 2 3], periodic BC in all three directions in 3D.

The first step in the generation of finite element matrices for periodic BC is to generate the matrices with Neumann BC along the boundary segments in the directions `p.sw.bcper`. Examples in the `OOPDE` setting are given in [RU18], and in 2D one can alternatively use the `Matlab pdetoolbox` function `assempde` and for rectangular geometries use the `pde2path` functions `recnbc*`.

The function `box2per` is then called to produce the transformation matrices and the new (reduced) number of uknowns `p.nu`. This is done in the call

$$[\texttt{p.mat.fill}, \texttt{p.mat.drop}, \texttt{p.nu}] = \texttt{getPerOp}(\texttt{p}, \texttt{dir}),$$

where `dir` is a local name for `p.sw.bcper`. Next, `box2per` drops the redundant entries in the solution vector:

$$\texttt{p.u} = [\texttt{p.mat.drop} * \texttt{p.u}(1 : \texttt{p.np} * \texttt{p.nc.neq}); \texttt{aux}],$$

where `aux` are the auxiliary variable in the solution vector (as described in the main manual). Finally, `box2per` transforms the finite element matrices by envoking `p=setfemops(p)`.

In `getPerOp` the product (3) of the one-directional transformation matrices is built. The matrices for each direction are built in `getPerOp1dir`. The transformation (2) of finite element matrices is carried out in `filltrafo`. Note that `filltrafo` is always called in `setfemops` resp. `oosetfemops` but because the `fill` and `drop` matrices are set to the scalar value 1 in `stanparam` by default, the transform is an identity unless `box2per` has been called. If (in $d \geq 2$) there are nonperiodic BC other than homogeneous Neumann, encoded via $Q_{\rm BC}$ and $G_{\rm BC}$, then we also need to transform these matrices, see §3.2.

When solution data are saved in a file (`pt*.mat`), the matrix data `p.mat` are not saved in order to limit disk space use. Instead, `p.mat.fill` and `p.mat.drop` are saved as empty matrices [ ] if periodic BC are used and as the scalar 1 otherwise. When loading data via `loadp`, the matrices `p.mat.fill` and `p.mat.drop` are built again via `getPerOp`.

3 Examples

3.1 pBC in 1D

For translationally invariant PDEs (such as (1)), pBC often (in 1D always) yield translationally invariant systems, which is problematic from the continuation point of view since nontrivial spatial derivatives of a solution u are then in the kernel of $\partial_u G(u)$. This phase invariance typically has to be fixed by a phase condition, see, e.g., [DRUW14, §2.5] and [RU17]. For simplicity, for the 1D case we consider a variant of (1) with some x–dependent terms (which break the translational invariance), namely

$$G(u) := -\mathrm{div}(c(x)\nabla u) - \lambda u - u^3 + \gamma u^5 - 0.5xu \stackrel{!}{=} 0, \quad c(x) = 1 + 0.1x^2, \tag{4}$$

on $\Omega = (-5, 5)$ with pBC, i.e., $u(5) = u(-5)$. We first fix $\gamma = 1$, and take λ as the primary bifurcation parameter. There is the trivial solution branch $u \equiv 0$, on which we find a number of bifurcation points. We follow the bifurcating branches, which show some folds, which we then continue in the second parameter γ. Finally we illustrate adaptive mesh refinement for this problem.

(4) has also been considered in [RU18, §3.3] (demo directory `acsuite/ac1Dxa`) with homogeneous Neumann BC, and we only need a few adaptions of these files. However, for convenience we give somewhat complete listings of the pertinent functions, summarized in Table 1.

Table 1: Functions in `ac1Dpbc`.

function	purpose,remarks
p=acinit(p,lx,nx,par)	init function, setting the domain, the grid and FEM operators, the parameters, and the starting point $u \equiv 0$ for the continuation.
p=oosetfemops(p)	set FEM matrices (stiffness K and mass M)
r=sG(p,u)	encodes G from 1 (including the BC)
Gu=sGjac(p,u)	Jacobian $\partial_u G(u)$ of G
Guuphi=spjac(p,u)	$\partial_u(\partial_u G(u)\phi)$, needed for fold continuation, see [RU18, §3.1.2]
[p,idx]=e2rs(p,u)	classical elements2refine selector as in pdejmps

The first two essential changes compared to the files in `ac1Dxa` occur in `acinit` and `oosetfemops`, namely

```
function p=acinit(p,lx,nx,par)  % init routine for AC on interval with pBC
p=stanparam(p); screenlayout(p); p.sw.sfem=-1;
p.fuha.sG=@sG; p.fuha.sGjac=@sGjac; p.fuha.e2rs=@e2rs; % the relevant fun.handles
pde=stanpdeo1D(lx,2*lx/nx); p.pdeo=pde; % domain and mesh
p.np=pde.grid.nPoints; p.nu=p.np; p.sol.xi=1/(p.nu); [po,t,e]=getpte(p);
p.mesh.bp=po; p.mesh.bt=t; p.mesh.be=e; % background mesh (for mesh adaption)
```

```
p.u=zeros(p.np,1); p.u=[p.u; par']; % initial guess (here 0, explicitly known)
p.sw.bcper=1; p=box2per(p); % prepare fill, drop for periodic BC, here in x
% p=setfemops(p); % would be called here in a non-periodic setting, now omitted
p.nc.nsteps=20; p.sw.foldcheck=1; p.plot.auxdict={'c','lambda','gamma','d'};
p.plot.pstyle=1; p.usrlam=[0 0.5 1]; p.nc.nsteps=100; p.sw.jac=1;
p.sw.bifcheck=2; p.nc.ilam=2; p.nc.lammax=2; p.sol.ds=0.1; p.nc.dsmax=0.2;
```

Listing 1: `ac1Dpbc/acinit.m`, very similar to `acsuite/ac1Dxa/acinit`. The difference is that in line 8 we transform the problem to a periodic domain via `p=box2per(p,1)`, i.e., compute `p.mat.fill` and `p.mat.drop`, and reduce the number of unknowns by 1. Then, since `setfemops` (which immediately calls `oosetfemops` due to `sfem=-1`) is already called in `box2per`, a call to `setfemops` can be omitted here.

```
function p=oosetfemops(p) % for 1Dpbc, with x-dep. K
x=getpte(p); c=1+0.1*x.^2; [K,M,~]=p.pdeo.fem.assema(p.pdeo.grid,c,1,1);
p.mat.M0=p.mat.fill'*M; % we need M0 to transform the nonlinearity
p.mat.K=filltrafo(p,K); p.mat.M=filltrafo(p,M); % transform of K and M
```

Listing 2: `ac1Dpbc/oosetfemops.m`. Lines 4,5 contain the transformation of the system matrices to the periodic domain.

`p.mat.fill` and its transpose are then used in `sG` to first extend u. Then f is computed on the extended grid and afterwards mapped back to the reduced grid, and the same applies to $\partial_u f$ in `sGjac` and $\partial_u(\partial_u G\phi)$ in `spjac` for fold continuation, see Listings 2-5.

```
function r=sG(p,u)  % AC with periodic BC
par=u(p.nu+1:end); up=u(1:p.nu); % params, and u on periodic domain
u=p.mat.fill*up; % extend ('fill') u to full domain
x=getpte(p); x=x'; % extract the point coordinates from p
f=par(2)*u+u.^3-par(3)*u.^5+0.5*x.*u; % f, with x-dependent term
F=p.mat.M0*f; % multiply by M, map back to active nodes of periodic domain
r=p.mat.K*up-F;  % bulk part of PDE
```

Listing 3: `ac1Dpbc/sG.m`. As u is the reduced solution (periodic boundaries dropped), it is first extended to the full domain where we compute the nonlinearity f (line 5), which is then mapped back to the periodic domain via M0=fill'*M, where M is the full mass matrix. On the other hand, the matrices K, Q in line 7 are already reduced, see Listing 2, and thus act on the reduced vector `up`.

```
function Gu=sGjac(p,u)  % PDE Jacobian for AC with pBC
par=u(p.nu+1:end); up=u(1:p.nu); % params, and u on periodic domain
u=p.mat.fill*up; % extend ('fill') u to full domain
x=getpte(p); x=x'; fu=par(2)+3*u.^2-5*par(3)*u.^4+0.5*x; % f_u on ext. domain
Fu=p.mat.M0*(spdiags(fu,0,p.np,p.np)*p.mat.fill); % map fu to per.dom
Gu=p.mat.K-Fu;
```

Listing 4: `ac1Dpbc/sGjac.m`. Jacobian of `sG.m` in Listing 3. Again, u is first extended to the full domain, where $\partial_u f$ is computed, which is then mapped back to the periodic domain via M0.

```
function Guuphi=spjac(p,u) % pa_u (G_u phi), called in getGu if p.spcontsw==1
nu=p.nu; n=p.np; par=u(2*nu+1:end);
phip=u(nu+1:2*nu); up=u(1:nu); % params, Evec, PDE-vars (per.domain)
phi=p.mat.fill*phip; u=p.mat.fill*up; % u and phi on extended domain
fuu=6*u-20*par(3)*u.^3; % 2nd derivative
Guuphi=-p.mat.M0*spdiags(fuu.*phi,0,n,n)*p.mat.fill; % mapped back to per.domian
```

Listing 5: `ac1Dpbc/spjac.m`, following the same principles as `sGjac` in Listing 4.

For some results from running `cmds.m` we refer to Fig. 3, which should also be compared to [RU18, Fig. 7]. At the end of `cmds.m` we also give an example of mesh-adaption with pBC, which in 1D works just as with any other type of BC.

```
%% cell 1: init and cont of trivial branch
p=[]; par=[1 -2 1 0.1]; % here par(4)=coefficient of x-dependent terms
p=acinit(p,5,80,par); p=setfn(p,'tr'); p=cont(p);
%% cell 2: switch to first 3 bifurcating branches and continue
p=swibra('tr','bpt1','b1',-0.1); p=cont(p);
```

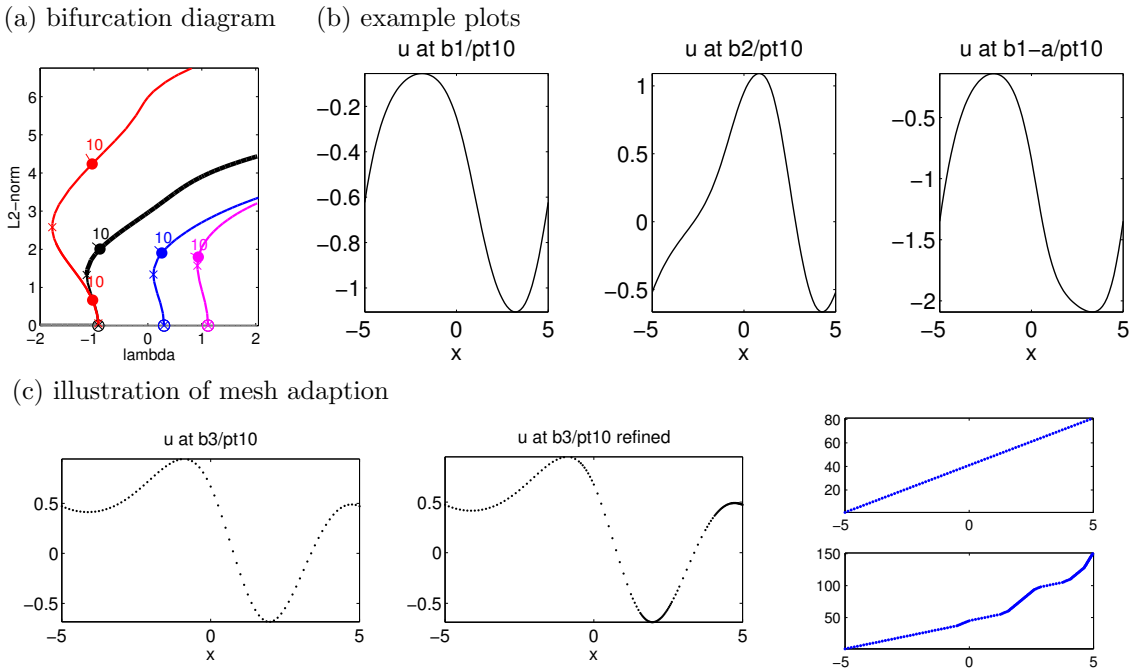

Figure 3: Results for (4) from `ac1Dpbc/cmds.m`. (a) Bifurcation diagram, (b) example plots from b1 (black in (a)), b2 (blue) and b1-a (red). (c) Example plots from mesh-adaption of b3/pt10 (magenta branch in (a)). Left: solution on original (equi-distributed) grid of 80 points. Middle: after grid adaption to 150 points. Right: point distribution before (top) and after (bottom) adaption. The default error estimator typically selects elements with high curvature for refinement, and it also sees the discontinuity of the term $0.5x\,u$ at the boundary of the periodic domain.

```
%% cell 3: fold-cont
p=spcontini('b1','fpt2',3,'b1f'); % init fold continuation with par 3 new prim.par
p.plot.bpcmp=p.nc.ilam(2); figure(2); clf; % use this new parameter for plotting
p.sol.ds=-0.01; p.nc.lammin=0.25; % new stepsize and range in new primary par.
p.sw.spjac=1; p.fuha.spjac=@spjac; % spectral jac
tic; p=cont(p,15); toc
%% cell 4: switch back to regular continuation from one of the fold points
p=spcontexit('b1f','pt13','b1-a'); p.nc.dsmax=0.5; p.sw.bifcheck=0; p.plot.bpcmp
    =0;
p.nc.lammin=-3; p.sol.ds=-1e-3; clf(2); p=cont(p,1); p=cont(p,20); % continue
p=loadp('b1-a','pt1','b1-b'); p.sol.ds=-p.sol.ds/5; % other direction
p.plot.bpcmp=0; p.nc.lammin=-3; p.nc.dsmax=0.3; p=cont(p,20);
%% cell 5: bifurcation diagram and solution plotting
f=3; c=0; figure(f); clf; % f=figure-Nr, c=component number (of branch)
plotbra('tr',f,c,'cl',[0.5 0.5 0.5],'lsw',0); plotbra('b1',f,c,'cl','k','lab',10);
```

Listing 6: `ac1Dpbc/cmds.m` (slightly abbreviated). Quite similar to `acsuite/ac1Dxa/cmds.m`. After init we first continue the trivial branch, with BP detection, and then follow the first three bifurcating branches (Cells 1 and 2). In Cell 3 we continue the left fold point on b1 in γ down to $\gamma = 0.25$, and in cell 4 we switch back to regular continuation in λ from this point again. Cells 5 and 6 deal with plotting, and then illustrate mesh adaption, which in 1D with pBC works as with other BC by computing the error estimates on the extended domain, see Listing 7.

```
function [p,idx]=e2rs(p,u)  % classical elements2refine selector as in pdejmps
par=u(p.nu+1:end);  a=0; [x,t]=getpte(p); x=x'; c=1+0.1*x.^2; % c and f on ext.dom
u=p.mat.fill*u(1:p.nu); fv=par(2)*u+u.^3-par(3)*u.^5+0.5*x.*u;
E=p.pdeo.errorInd(u,c,a,fv);
p.sol.err=max(max(E)); idx=p.pdeo.selectElements2Refine(E,p.nc.sig);
```

Listing 7: `ac1Dpbc/e2rs.m`. For the error estmates we set $a = 0$ and compute c, f on the extended domain.

3.2 pBC in 2D

In 2D we consider (1) on the rectangle $\Omega = (-2\pi, 2\pi) \times (-\pi, \pi)$, i.e.,

$$-c\Delta u - \lambda u - u^3 + \gamma u^5 = 0 \quad \text{in } \Omega, \tag{5}$$

$$\text{with BC } u = 0 \text{ on } x = -2\pi,\ u = d\sin(y+1) \text{ on } x = 2\pi, \text{ and periodic BC in } y. \tag{6}$$

Note that neither homogeneous Dirichlet nor Neumann BC on $y = \pm\pi$ are compatible with the BC on $x = 2\pi$. On the other hand, the y–dependent BC on $x = 2\pi$ in (6) have the effect that the problem is not translationally invariant in y.

We adapt files from `acsuite/ac2D`, and refer to [RU18] for the implemntation of the BC on $x = 2\pi$ via a matrix Q_{BC} and a vector G_{BC}. Then again the first two pertinent changes compared to the files in `acsuite/ac2D` occur in `acinit` and `oosetfemops`, and afterwards the changes in `sG, sGjac` and `spjac` follow the same principles as in §3.1.

```
function p=acinit(p,lx,ly,nx,par)
p=stanparam(p); screenlayout(p); p.sw.sfem=-1;
p.fuha.sG=@sG; p.fuha.sGjac=@sGjac; p.fuha.e2rs=@e2rs;
pde=stanpdeo2D(lx,ly,2*lx/nx); % % domain and mesh, h as argument
p.pdeo=pde; p.np=pde.grid.nPoints; p.nu=p.np; p.sol.xi=1/(p.nu);
p.u=zeros(p.np,1); p.u=[p.u; par']; p.nc.nsteps=20;
p=box2per(p,2); % prepare fill, drop for periodic BC, here in y
```

Listing 8: `ac2Dpbc/acinit.m`. Line 7 (and the subsequent omission of `setfemops`) contains the only change compared to `acsuite/ac2D/acinit.m`.

```
function p=oosetfemops(p) % in problem-dir
gr=p.pdeo.grid; [K,M,~]=p.pdeo.fem.assema(gr,1,1,1); % indep. of BC
bc1=gr.robinBC(0,0); bc2=gr.robinBC(1,'sin(y+1)'); bc4=gr.robinBC(1,0);
gr.makeBoundaryMatrix(bc1,bc2,bc1,bc4); % bottom, right, top, left
[Q,G,~,~]=p.pdeo.fem.assemb(gr);  % the BC matrices
p.nc.sf=1e3; % stiff spring constant for DBC via Robin-BC
p.mat.M0=p.mat.fill'*M; % we need M0 to transform the nonlinearity
p.mat.K=filltrafo(p,K); p.mat.M=filltrafo(p,M); % standard transforms of
p.mat.Q=filltrafo(p,Q); p.mat.G=p.mat.fill'*G; % system matrices
```

Listing 9: `ac2Dpbc/oosetfemops.m`. Uncomment line 4 to identify the boundary segments. Here the Dirichlet BC are on segments 2 and 4, they are implemented by the stiff-spring approximation with stiff spring constant `p.nc.sf` (set in line 6), and the associated boundary matrices/vectors are set up in lines 7–9. Afterwards, all the system matrices, including Q and G are transformed to the periodic domain.

For mesh refinement in 2D with pBC we need to adapt `e2rs.m`, see Listing 10. The selection of elements to refine happens on the full (extended) domain, but in in order to keep the identification of periodic boundaries consistent we remove boundary triangles from the refinement list, using the function `rmbdtri.m`.

```
function [p,idx]=e2rs(p,u)  % classical elements2refine selector as in pdejmps
par=u(p.nu+1:end); c=par(1); a=0; fv=nodalf(p,u);
u=p.mat.fill*u(1:p.nu); E=p.pdeo.errorInd(u,c,a,fv);
p.sol.err=max(max(E)); idx=p.pdeo.selectElements2Refine(E,p.nc.sig);
idx=rmbdtri(p,idx); % rm triangles near per.bdry from ref.list
```

Listing 10: `ac2Dpbc/e2rs.m`; the error is estimated on the full domain, but elements at the periodic boundaries afterwards need to be removed from the refinement list.

With these preparations we can now proceed as in demo `acsuite/ac2D`; starting from $d = 0$ and $u \equiv 0$ we first continue in d, and then in λ to detect and localize the bifurcation points from the 'primary' branch. See `ac2Dpbc/cmds.m` for the code, which also contains a fold continuation, and Fig. 4 for some example results.

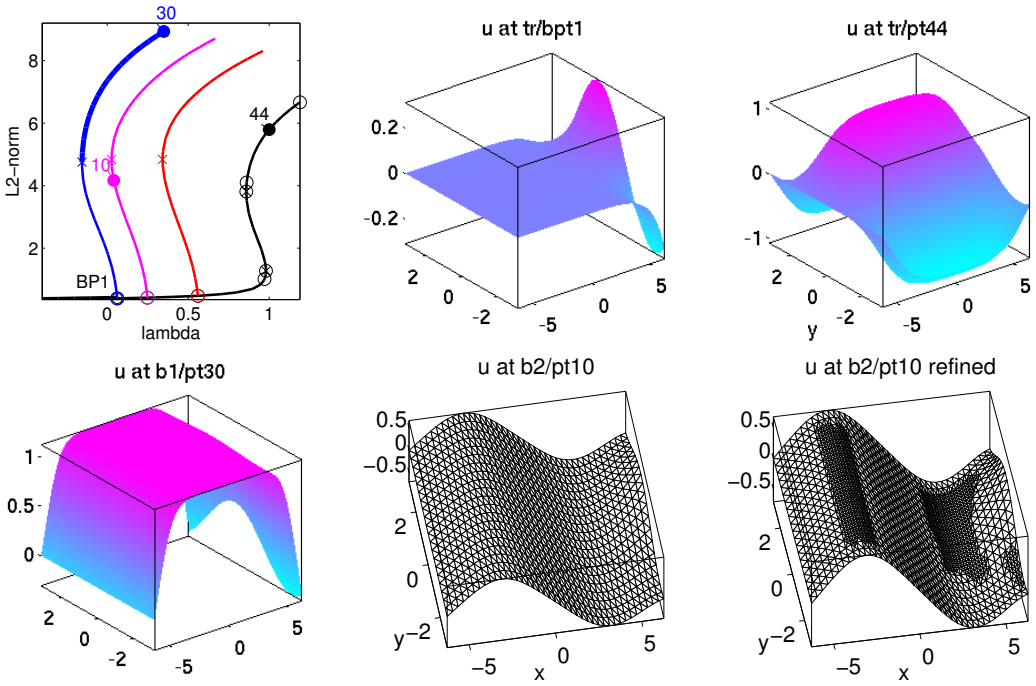

Figure 4: Example results for (5), (6). Primary branch (`tr`, black) and 3 bifurcating branches (`b1`, blue; `b2`, magenta, and `b3` red). In the last two plots we illustrate mesh-adaption at b2/pt10. The error-estimator mainly selects triangles with high curvature, but triangles near the y-boundaries are removed from the refinement list via `rmbdtri`.

3.3 pBC in 3D

To give a 3D example with periodicity in all three space directions, we break the translational invariance of (1) in all directions by adding the term $h(x, y, z)u$ to (1), i.e., we consider

$$-c\Delta u - \lambda u - u^3 + \gamma u^5 - h(x, y, z)u \stackrel{!}{=} 0, \tag{7}$$

where we choose

$$h(x, y, z) = 1/(1 + (x + 1)^2 + (y + 1)^2 + (z + 1)^2).$$

See `sG.m` and `sGjac.m` in `acpbc3D` for this modification of `acsuite/ac3D`, where we use some (non-homogeneous) Dirichlet BC. We run the model on the same domain as in [RU18, §5], namely $\Omega = (-2\pi, 2\pi) \times (-3\pi/2, 3\pi/2) \times (-\pi, \pi)$, and thus the only relevant modifications concern the boundary conditions. We call `p=box2per(p,[1 2 3])` in `acinit.m`, and due to the periodicity in all 3 directions we do not set any BC in `oosetfemops`. See Fig. 5 for some example results, which also illustrate that in 3D it is useful to play with different plot options. Please also explore some other branches for interesting solutions/pictures by using suitable modifications in `cmds.m`.

3.4 A quasilinear problem with periodic BC

In [RU18, §5] we consider a quasilinear variant of (1), namely

$$-\nabla \cdot [c(u)\nabla u] - f(u) = 0, \tag{8}$$

with $c(u) = c_0 + \delta u + \varepsilon u^2$, and $f(u, \lambda) = \lambda u + u^3 - u^5$, in 1D, 2D and 3D. Here we focus on 2D with pBC in x and y direction, and to break translational invariance, similar to (7), change f to $f(u, \lambda) = \lambda(x)u + u^3 - u^5$ where $\lambda(x) = \dfrac{\lambda}{1 + 0.5((x + 1)^2 + (y + 1)^2)}$. The FEM formulation of (8) reads

$$G(u) := K(u)u - F(u) = 0, \tag{9}$$

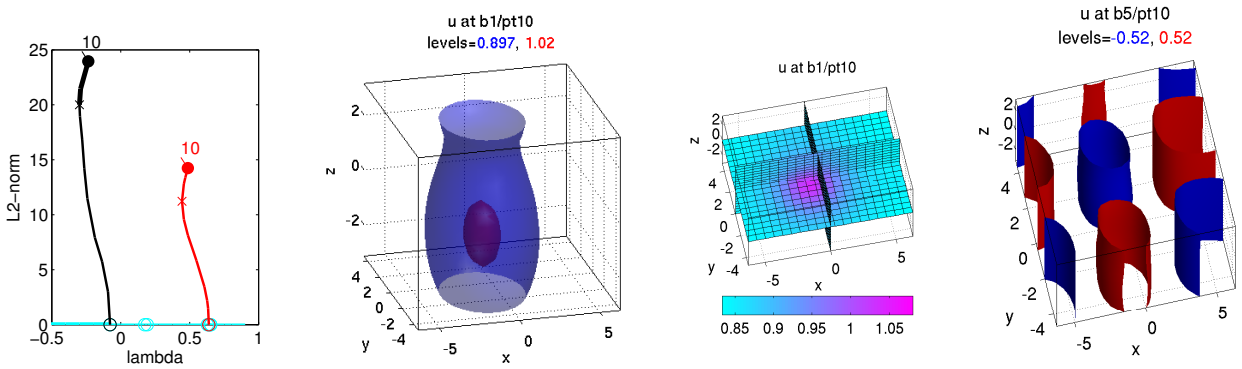

Figure 5: Example outputs from `ac3Dpbc/cmds.m`.

respectively $M\dot{u} = -G(u)$ for the evolutionary problem, where now

$$K_{ij}(u) = \int_\Omega c(u)\nabla\phi_i \cdot \nabla\phi_j \, \mathrm{d}x \tag{10}$$

depends on u. This needs $c(u)$ on the element centers, which is obtained from first interpolating u from the nodes to the element centers. Similarly, instead of using Mf, which is also possible and up to a $\mathcal{O}(h^2)$ interpolation error gives the same results, see [RU18, Remark 1.1], we now also evaluate F via `assema`. As usual, for pBC we need to combine this with first extending u to the full domain, and then reducing the assembled matrices/vectors to the reduced domain, for which we need `fill, drop`, generated in `oosetfemops`. Additionally, in `oosetfemops` we also generate and save the space-dependent factor `xft` for λ, and a matrix `p2c`, which in `OOPDE` yields the interpolation from point values to element centers. Then we can encode G as in `sG` in Listing 11.

```
function r=sG(p,u)  % rhs for ql-AC
par=u(p.nu+1:end); c0=par(1); lam=par(2); ga=par(3); del=par(4); epsi=par(5);
u=u(1:p.nu); uf=p.mat.fill*u; ut=(p.mat.p2c*uf)'; % interpol. to elem. centers
c=c0+del*ut+epsi*ut.^2; f=lam*p.xft.*ut+ut.^3-ga*ut.^5;
[K,~,F]=p.pdeo.fem.assema(p.pdeo.grid,c,0,f); % assemble K and F (M not used)
K=filltrafo(p,K); F=p.mat.fill'*F; r=K*u-F;
```

Listing 11: `acqlpbc/sG.m`, for (8) with periodic BC. We now need to assemble $K = K(u)$ in each step. In line 4 we interpolate `uf` to the element centers, in line 5 compute c and f, and in line 6 we assemble K and F; remainder as usual.

The main trick to encode the Jacobian

$$G_u(u)v = -\nabla \cdot (c(u)\nabla v) - \nabla \cdot (c_u(u)\nabla uv) - f_u(u)v, \tag{11}$$

is to use differentiation matrices `p.Dx` and `p.Dy`, also generated in `oosetfemops`, to obtain the coefficients u_x, u_y needed in (11). These act on the full vector, and hence are not transformed via `filltrafo`. The first order terms $\nabla\cdot(c_u(u)\nabla uv)$ are then approximated via the matrix `K1` in `sGjac2D`, see Listing 12.

```
function Gu=sGjac2D(p,u)  % Jac for ql-AC
par=u(p.nu+1:end); c0=par(1); lam=par(2); ga=par(3); del=par(4); epsi=par(5);
n=p.nu; u=u(1:n); uf=p.mat.fill*u; M=p.mat.M; gr=p.pdeo.grid;
ut=(p.mat.p2c*uf)'; c=c0+del*ut+epsi*ut.^2; % diff. coefficient defined on centers
cu=del+2*epsi*uf; fu=lam*p.xft+3*ut.^2-5*ga*ut.^4;
[K,Fu,~]=p.pdeo.fem.assema(gr,c,fu,0); Fu=filltrafo(p,Fu);  K=filltrafo(p,K);
ux=p.mat.Dx*uf; uy=p.mat.Dy*uf; % 1st derivatives as coefficients
cuux=filltrafo(p,spdiags(cu.*ux,0,p.np,p.np)); % coeff.matrix
cuuy=filltrafo(p,spdiags(cu.*uy,0,p.np,p.np));
K1=p.mat.Kx*cuux+p.mat.Ky*cuuy; % first order derivatives acting on v
Gu=K-K1-Fu;  % putting it all together
```

Listing 12: `acqlpbc/qlsGjac.m`, implementing (11) with periodic BC. In line 6 we assemble `Fu`, together with `K`, and then we set up the first order terms from (11).

With these preparations we may now proceed as in §3.2 to continue the primary branch and follow some bifurcations. Mesh adaption works as before.

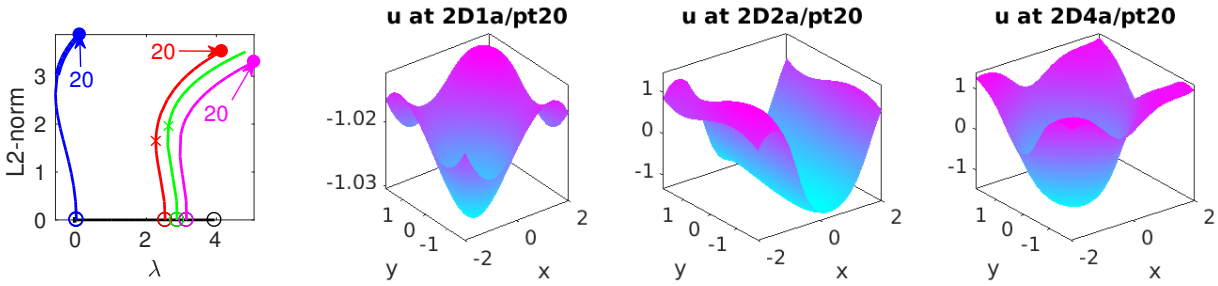

Figure 6: Example outputs from `acqlpbc/cmds2D.m`.

References

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

[DU16] T. Dohnal and H. Uecker. Bifurcation of Nonlinear Bloch waves from the spectrum in the nonlinear Gross-Pitaevskii equation. *J. Nonlinear Sci.*, 26(3):581–618, 2016.

[dWDR+18] H. de Witt, T. Dohnal, J. Rademacher, H. Uecker, and D. Wetzel. pde2path - Quickstart guide and reference card, 2018. Available at [Uec18].

[RU17] J. Rademacher and H. Uecker. Symmetries, freezing, and Hopf bifurcations of modulated traveling waves in pde2path, 2017.

[RU18] J. Rademacher and H. Uecker. The OOPDE setting of pde2path – a tutorial via some Allen-Cahn models, 2018.

[Uec18] H. Uecker. `www.staff.uni-oldenburg.de/hannes.uecker/pde2path`, 2018.

[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.

[ZHKR15] D. Zhelyazov, D. Han-Kwan, and J. D. M. Rademacher. Global stability and local bifurcations in a two-fluid model for tokamak plasma. *SIAM J. Appl. Dyn. Syst.*, 14(2):730–763, 2015.