pde2path–V2: multi-parameter continuation and periodic domains, 28.3.2014

Tomáš Dohnal[1], Jens D.M. Rademacher[2], Hannes Uecker[3], Daniel Wetzel[3]
[1] *TU Dortmund, Fakultät für Mathematik, 44227 Dortmund,* [2] *Universität Bremen, Fachbereich Mathematik, 28359 Bremen,* [3] *Universität Oldenburg, Institut für Mathematik, 26111 Oldenburg*

Summary. p2p2 is an upgrade of the continuation/bifurcation package pde2path for elliptic systems of PDEs, based on Matlab's pdetoolbox. The new features include easier switching between different single parameter continuations, genuine multi–parameter continuation (e.g., fold continuation), cylinder and torus geometries (i.e., periodic boundary conditions), a general interface for adding auxiliary equations like mass conservation or phase equations for continuation of traveling waves, and a more efficient FEM usage.
MSC: 35J47, 35J60, 35B22, 65N30
Keywords: elliptic systems, continuation and bifurcation, finite element method

Introduction

pde2path, based on the FEM of the Matlab pdetoolbox, is a continuation/bifurcation package for elliptic systems of PDEs of the form

$$G(u,\lambda) := -\nabla\cdot(c\otimes\nabla u) + au - b\otimes\nabla u - f = 0, \tag{1}$$

where $u = u(x) \in \mathbb{R}^N$, $x \in \Omega \subset \mathbb{R}^2$ some bounded domain, $\lambda \in \mathbb{R}^p$ is a parameter (vector), $c \in \mathbb{R}^{N\times N\times 2\times 2}$, $b \in \mathbb{R}^{N\times N\times 2}$, $a \in \mathbb{R}^{N\times N}$ and $f \in \mathbb{R}^N$ can depend on $x, u, \nabla u$, and, of course, parameters. For the basic ideas of continuation/bifurcation, the algorithms, and the class of systems we aim at, i.e., the meaning of the terms in (1) and the associated boundary conditions, we refer to [7], and the references therein. Here we explain a number of additional features in pde2path V2, in short p2p2, compared to the version documented in [7], and some changes in the underlying data structures. The software can be downloaded from [3], together with a number of demo–directories and a detailed manual. The new features include:

1. easy switching between different single parameter continuations;
2. genuine multi–parameter continuation, in particular automatic fold and branch point continuation;
3. general interface for adding auxiliary equations, such as mass conservation, or phase equations for continuation of traveling waves;
4. periodic domains: cylinder and torus geometries;
5. fast FEM assembling for a subclass of (1), roughly where c, a, b are independent of u;
6. improved and more user-friendly plotting.

We explain these features by a number of examples, but first we describe the major structural changes.

Remark 1. The new data structure and different user interfaces mean that there is no downward compatibility with [7]. On the other hand, we think that upgrading old pde2path–files to p2p2 is quickly achieved, and that the data structure and user interfaces now have a final form.

New data structure and user interfaces.
A p2p2 problem is described by a matlab structure p. The most drastic change compared to [7] is that no single distinguished parameter λ appears in p anymore, but any number of auxiliary variables, typically parameters, can be added. If the FEM mesh has np points and N =neq in (1) we have p.nu=neq*p.np unknown nodal values for u, (except in case of periodic BC, see below), and p.u(1:p.nu) contains these nodal values. The arbitrary number of auxiliary variables are stored in p.u(p.nu+1:end) and can be "passive", serving as constant parameters, or "active" unknowns to be solved for. In the following we write, on the discrete level, $U = (u, w)$ =p.u, where u corresponds to the (nodal values of) the PDE variables in (1) and w the auxiliary variables. Suppose there are $n_q + 1$ active variables $w_{\rm act} \in \mathbb{R}^{n_q+1}$. Exactly one of these is the "primary" active parameter, and we write $w_{\rm act} = (\tilde{w}, \alpha)$. The remaining n_q active variables require n_q additional ('auxiliary') equations

$$q_i(U) = 0, \quad i = 1, \ldots, n_q. \tag{2}$$

In the functions defining G or its Jacobian a typical first step is to split off the PDE part u as shown in the examples below. The active auxiliary variables are selected by the user in the array of indices p.nc.ilam, whose first entry is the primary continuation parameter. For different continuation tasks the user may freely modify this list to choose different active, passive and primary parameters. Thus, $w_{\rm act} = (\tilde{w}, \alpha)$ is only a symbolic notation, and the role of parameters (primary, active, passive) is determined by p.nc.ilam. Examples of additional equations are:

- prescribed mass: $\int u\,\mathrm{d}x - m = 0$, $m \in \mathbb{R}$; see (11b);
- a phase condition $\langle \partial_x u, u - u_{\rm old}\rangle_2 = 0$ for the continuation of travelling waves.

As discussed in [7], it is useful to give u and the continuation parameter different weights in the arclength equation

$$p(U,s) = \left\langle \dot{U}, U(s) - U_0 \right\rangle - (s - s_0) = 0; \tag{3}$$

see [7, §2.1]. In `p2p2` this is extended to the active variables in $w_{\rm act} = (\tilde{w}, \alpha)$, and as scalar product in (3) we use,

$$\langle (u, \tilde{w}, \alpha), (v, \tilde{z}, \beta) \rangle := \xi \langle u, v \rangle_2 + \xi_q \langle \tilde{w}, \tilde{z} \rangle_2 + (1 - (\xi + \xi_q)/2)\alpha\beta, \tag{4}$$

with independent weights ξ and ξ_q and $\langle \cdot, \cdot \rangle_2$ the euclidean inner product.
In order to ease switching between different primary parameters, and since finite difference approximations of derivatives of G with respect to just a few parameters are relatively cheap, we also deleted all explicit references to $\partial_\lambda G$, i.e., derivatives of G with respect to the active parameters are now done only numerically. Hence, the interfaces for the functions defining $G(u)$ and its Jacobian now read `function [c,a,f,b]=G(p,u)` and `[cj,aj,bj]=Gjac(p,u)`, see the examples below.
Finally, in `p2p2` the many switches and settings in the problem structure variable `p` in [7] are now grouped as explained in Table 1. In particular, this makes it easier to get an overview over current parameter settings. For instance, to see the values of the numerical control parameters for a given `p`, type `p.nc` at the command line.

field	purpose
fuha	struct of **fu**nction **ha**ndles in particular the function handles p.fuha.G, p.fuha.Gjac, p.fuha.bc, p.fuha.bcjac defining (1) and Jacobians, but also others such as p.fuha.outfu, p.fuha.savefu, ...
nc	struct of **n**umerical **c**ontrols such as p.nc.tol, ...
sw	struct of **sw**itches such as p.sw.bifcheck,...
u,np,nu	the solution u(1:p.nu+p.nc.nq+1), and the number of nodes p.np in the mesh, and nodal values p.nu of PDE–variables,
tau,branch	tangent tau(1:p.nu+p.nc.nq+1), and the branch, filled via bradat.m and p.fuha.outfu.
sol	other values/fields calculated at runtime
eqn	struct containing the tensors c, a, b for the simple FEM setup, see the examples.
mesh	the geometry data and mesh.
plot	switches (and, e.g., figure numbers) for plotting
file	switches and presets for file output
time	timing information
pm	pmcont switches
fsol	switches for the interface to `fsolve`, see Remark 2.
mat	problem matrices, e.g., mass/stiffness matrices K, M for the the simple FEM setting, and `drop` and `fill` for pBC; by default, the struct mat is *not* saved to disk, see also Remark 4.

Table 1: Main fields in the structure `p` describing a p2p2 proplem; see `stanparam.m` in `p2plib` for detailed information on the contents of these fields and the standard settings. The destinction between `nc` and `sw` is somewhat fuzzy, as both contain variables to control the behaviour of the numerics: the rule is that `nc` contains numerical constants, real or integer, while the switches in `sw` only take a finite numbers of values like 0,1,2,3. Finally, `u,np,nu,tau` and `branch` are *not* grouped into a substructure, as in our experience these are the variables most often accessed directly by the user.

Remark 2. Concerning the improved plotting, `p2p2` uses telling axes labelling and, for instance, a simplified user-friendly branch-plotting command: `plotbra(p)`. By default, this plots the branch with the primary parameter on the x-axis and the L^2-norm (now stored in the internal part of the branch data) on the y-axis; the figure used can be controlled by `p.plot.brafig`. Similarly, `plotbraf('p')` is now allowed for convenience and calls `plotbra(p)` with structure `p` from the file in directory `'p'` with the highest point label. Finally, we also added a wrapper to call `Matlab`'s `fsolve` routine; although this is typically slower than our own Newton loops, it may be useful, for instance, to find solutions from poor initial guesses, see the demo `fCH`.

Examples

The Allen-Cahn model (demos `acfold` and `acfront`).
As a first example we (re)consider the cubic–quintic Allen-Cahn equation from [7, §3.2], written as

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

on the rectangle $\Omega = [-1, 1] \times [-0.9, 0.9]$ with homogeneous Dirichlet BC. Our goal is to illustrate the new meaning of `p.u`, parameter–switching and fold–continuation, and a new setup with a more efficient usage of the FEM. The demo directory for this is `acfold`.
There are three parameters λ, c, γ, hence, in addition to the standard domain and BC setup known from [7], the init-routine now initializes those and sets the primary continuation parameter to λ.

```
% initialize auxiliary variables, here parameters of PDE
par(1)=1;    % linear cofficient of f
par(2)=0.25; % diffusion coefficient
par(3)=1;    % quintic coefficient of f
p.u=[p.u; par']; % augment p.u by parameters
p.nc.ilam=1; % set active parameter indices (here only one)
```

The functions defining G and its Jacobian read

```
function [c,a,f,b]=acG(p,u)  % coefficient functions for AC
% separate pde and auxiliary variables, here "par", and interpolate to triangle centers
par=u(p.nu+1:end); u=pdeintrp(p.points,p.tria,u(1:p.nu));
c=par(2); a=0; b=0; f=par(1)*u+u.^3-par(3)*u.^5; end
function [cj,aj,bj]=acGjac(p,u)  % jacobian for AC
par=u(p.nu+1:end); u=pdeintrp(p.points,p.tria,u(1:p.nu));
cj=par(2); bj=0; fu=par(1)+3*u.^2-par(3)*5*u.^4; aj=-fu; end
```

Remark 3. We recall, see [7, Remark 3.2], that `cj,aj,bj` in `Gjac` are *not* the derivatives of c, a, b in G. The notation only indicates that `cj,aj,bj` are the coefficients needed to assemble G_u. In general, the relation between `cj,aj,bj` and c, a, b, f can be quite complicated, and only if c, a, b are independent of u, and f only depends on u without derivatives (roughly: the semilinear case), then `cj`$= c$, `bj`$= b$, and `aj`$= a - f_u$.

Efficient FEM usage.
Exploiting a semilinear structure in the FEM assembling can give a significant computational speedup: the FEM representation $G(u) = Ku - F$ of, e.g., $-\Delta u - f(u)$, can be obtained directly from $F =$`p.mat.M*f(u)+p.mat.bcG` and $Ku=$`p.mat.K*u`, where `p.mat.M` and `p.mat.K` are the mass and stiffness matrices, `f(u)` denotes $f(u)$ as nodal values, and `p.mat.bcG` comes from the boundary conditions. In contrast, the FEM assembling via the general routine `[c,a,f,b]=G(p,u)`, calculates the coefficients `c,a,f,b` on the (larger number of) triangles after interpolation, and then `K`, `F` are assembled from these at every Newton set of every continuation step.
In `p2p2` this 'simple' FEM assembling is turned on by `p.sw.sfem=1`, which requires implementing the nodal routines for the Jacobian and residual, as well as setting the divergence tensor `p.eqn.c` and, if needed, `p.eqn.a` and `p.eqn.b`. The matrices M and K are then generated via `p=setfemops(p)` and stored in `p.mat`. For the `acfold` demo the setup and relevant routines read:

```
p.sw.sfem=1; p.fuha.sG=@acsG; p.fuha.sGjac=@acsGjac; p.eqn.c=1;
function r=acsG(p,u)
par=u(p.nu+1:end); u=u(1:p.nu); f=par(1)*u+u.^3-par(3)*u.^5;
r=par(2)*p.mat.K*u(1:p.nu)-p.mat.M*f; end
function Gu=acsGjac(p,u)
par=u(p.nu+1:end); fu=par(1)+3*u.^2-par(3)*5*u.^4; Fu=spdiags(fu,0,p.nu,p.nu);
Gu=par(2)*p.mat.K-p.mat.M*Fu; end
```

For problems involving the advection tensor b, analogously use the matrix `p.mat.Kadv` in the routines; see the `acfront` and `schnaktravel` demos.

Remark 4. See [3] for more examples where the `sfem=1` setting applies. If such a problem is run on a fixed mesh, then `p.sw.sfem=1` and setting `p.fuha.sG` and `p.fuha.sGjac` as above can replace the old setting with `p.fuha.G` completely. However, if adaptive mesh refinement is desired, then `p.fuha.G` is still needed to identify the triangles to be refined. The needed new matrices `p.mat.M`, `p.mat.K` and `p.mat.Kadv` are automatically reassembled during mesh adaption. Moreover, to save hard disk space, the field `p.mat` is not saved. When loading a point from file via `loadp`, `p.mat` is automatically regenerated. However, when loading a point `p` into the `Matlab` workspace by a double click, this is not the case and a manual call to `setfemops` is needed.

Remark 5. While `p.mat.bcG=0` in `acsG` and `acsGjac`, we caution the user that in the case of inhomogeneous boundary conditions the corresponding FEM terms need to be accounted for in the nodal implementation.

Fold detection, point types and parameter switching.
After locating the well-known bifurcation points (eigenvalues of the Dirichlet Laplacian) from the trivial branch $u \equiv 0$ we switch to the first bifurcating branch and continue it including fold-detection by

```
q=swibra('p','bp1','q',0.2); q.sw.foldcheck=1; q=cont(q);
```

where fold detection works by bisection as for branch points. The resulting branch is plotted in Figure 1(a) with the fold point marked (it is also stored in the file `q/fp1.mat`) and it is assigned a point type in the branch `p.branch`. Point types in `p2p2` branches are:

-1	= initial point or restart
-2	= guess from `swibra` for the initialization of branch switching
0	= normal point
1	= bifurcation point (found with `bifdetec`)
2	= fold point (found with `folddetec`)

A parameter switch to continue a stored solution in the previously passive diffusion rate parameter c (parameter number 2 in the implementation) goes simply by `w=swiparf('q','p10','w',2);` where the essential change done by `swiparf` is setting `w.nc.ilam=2;`. Before continuation by `w=cont(w);` some adjustments to the settings are useful in this case: `w.nc.lammin=0.1; w.sol.ds=-0.01; w.sol.xi=1e-6;` where the small weight ξ is useful since the problem is more sensitive in the diffusion coefficient.

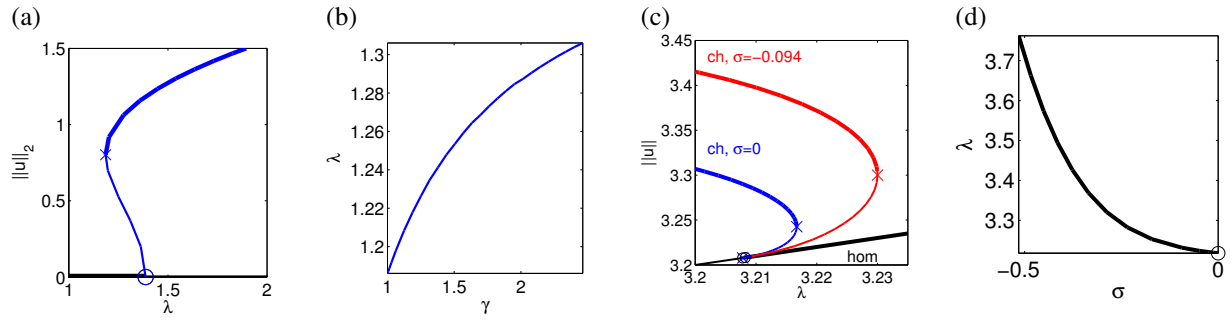

Figure 1: (a),(b) First bifurcating branch and fold–continuation in the Allen–Cahn model (5). (c) "cold hexagon" solution branch (blue) for (8) with σ=0. (d) continuation of the fold for σ=0 from (c) in σ; afterwards, the red branch in (c) was obtained via `foldexit` at $\sigma \approx -0.094$ and continuing in λ again, with positive and negative `ds`. See `schnakcmds.m` for details.

Fold and branch point continuation.

Constraining continuation to folds or branch points requires an additional free parameter. For the Allen–Cahn model we set `p.nc.ilam=[3,1]`, thus $w_{\text{act}} = (\lambda, \gamma)$, i.e., γ becomes the new primary parameter and λ (the fold position in the old primary parameter) will be calculated. In general, `p2p2` now discretizes the extended system

$$H(U) = \begin{pmatrix} G(u,w) \\ \partial_u G(u,w)\phi \\ \|\phi\|_{L^2}^2 - 1 \\ p(U) \end{pmatrix} = 0, \quad U = (u, \phi, w), \tag{6}$$

where ϕ is in the kernel of $\partial_u G$ with L^2-norm constrained to one by the third equation, and $p(U) = 0$ is the arclength equation (3). Thus (6) is a system of `p.nu+p.nu+2` equations in `p.nu+p.nu+2` unknowns. For continuation of (6) we need the Jacobian

$$D_U H(U) = \begin{pmatrix} \partial_u G & 0 & \partial_w G \\ \partial_u(\partial_u G\phi) & \partial_u G & \partial_w(\partial_u G\phi) \\ 0 & 2\phi^T & 0 \\ \xi\dot{u}^T & \xi\dot{\phi}^T & (1-(\xi+\xi_q)/2)\dot{w} \end{pmatrix}, \tag{7}$$

where $\partial_u G$ is assembled as usual (in the demo by `acsGjac`), ϕ only occurs linearly in (6), derivatives with respect to w are done via finite differences, and the computationally most costly part is the evaluation of $\partial_u(\partial_u G\phi)$. While this is done by default via finite differences, the user is urged to implement $\partial_u(\partial_u G\phi)$ in a routine `p.fuha.spjac` and set `p.sw.spjac=1`.
In the `acfold` demo, $\partial_u(\partial_u G\phi) = f_{uu}\phi$ so that we can use the pre-assembled mass matrix `p.mat.M` as in the simple FEM assembling discussed above:

```
function Guuph=acspjac(p,u)
ph=u(p.nu+1:2*p.nu); par=u(2*p.nu+1:length(u)); u=u(1:p.nu);
fuu=6*u-20*par(3)*u.^3; Guuph=-(p.mat.M*spdiags(fuu,0,p.nu,p.nu))*spdiags(ph,0,p.nu,p.nu);
```

The use of the extended system (6) and its Jacobian for subsequent continuations (the fold/branch point-continuation mode) in `p2p2` is turned on by calling `spcontini`; for instance:

```
qf=spcontini('q','fp1',3,'qf');% init fold continuation with par 3 as new active parameter
qf.plot.bpcmp=3; clf(2);       % use this new parameter for plotting
qf.sol.ds=1e-3;                % new initial stepsize in new primary parameter
```

Then calling `qf=cont(qf)` yields the branch plotted in Figure 1(b). Normal continuation from a point stored from a fold- or branch point-continuation is turned back on by calling `spcontexit` as in:

```
q1=spcontexit('qf','p10','q1'); q1.nc.tol=1e-8; q1.sol.ds=1e-3; q1=cont(q1);
```

Examples of branch point continuation are given in the demo `acfold` for the trivial state, and in the demo `bratu` for a nontrivial state.

Fold continuation in a system: the Schnakenberg model (demo `schnakfold`).

As an example for a system we consider the stationary Schakenberg model

$$G(U) := -D\Delta U - N(U,\lambda) - \sigma\left(u - \frac{1}{v}\right)^2 \begin{pmatrix} 1 \\ -1 \end{pmatrix} = 0, \quad N(U,\lambda) = \begin{pmatrix} -u + u^2 v \\ \lambda - u^2 v \end{pmatrix}, \tag{8}$$

with $U = (u,v)(x,y) \in \mathbb{R}^2$, diffusion matrix $D = \left(\begin{smallmatrix} 1 & 0 \\ 0 & d \end{smallmatrix}\right)$, d fixed to $d = 60$, and bifurcation parameters $\lambda \in \mathbb{R}_+$ and $\sigma \in \mathbb{R}$. (8) has the homogeneous solution $(u,v) = (\lambda, 1/\lambda)$, which becomes Turing unstable for $\lambda \le \lambda_c \approx 3.2085$,

independent of σ, with critical wave-vectors k with $|k| = k_c = \sqrt{\sqrt{2}-1}$. Here σ can be used to turn certain 2D bifurcations from sub–to supercritical, and many branches of patterns exhibit one or many folds ("snaking") [5].
We consider (8) on suitable rectangular domains with with homogeneous Neumann BC for both, u and v, with the focus on fold continuation. This can be used to discuss snaking widths, see [6], which however, requires rather large systems with meshes of $\mathcal{O}(10^5)$ many points, so that for efficiency reasons finite differences for $\partial_u(\partial_u G\phi)$ in (7) are not an option. Following the approach discussed above, here we implement $\partial_u(\partial_u G\phi)$ in the simplified nodal FEM format, which we also use for the PDE itself. Denoting $u = (u_1, u_2)$, $\phi = (\phi_1, \phi_2)$, $f = (f_1, f_2)$ for the components of u, ϕ and f, we have

$$\partial_u(\partial_u G\phi) = \begin{pmatrix} (\partial^2_{u_1} f_1)\phi_1 + (\partial_{u_1}\partial_{u_2} f_1)\phi_2 & (\partial_{u_1}\partial_{u_2} f_1)\phi_1 + (\partial^2_{u_2} f_1)\phi_2 \\ (\partial^2_{u_1} f_2)\phi_1 + (\partial_{u_1}\partial_{u_2} f_2)\phi_2 & (\partial_{u_1}\partial_{u_2} f_2)\phi_1 + (\partial^2_{u_2} f_2)\phi_2 \end{pmatrix}. \tag{9}$$

Using the nodal values for $\partial_{u_i}\partial_{u_j} f_k$ and multiplication with `p.mat.M`, this is implemented in `schnakspjac.m`. The continuation in `p2p2` works as discussed in the previous sections, where now we continue the fold position λ in the new primary parameter σ. See Fig. 1(c),(d) for example results.

Integral constraints: the functionalized Cahn-Hilliard equation (demo `fCH`)
As an example with additional equations we consider the functionalized Cahn–Hilliard equation from [1],

$$\partial_t u = -\mathcal{G}[(\varepsilon^2\Delta - W''(u) + \varepsilon\eta_1)(\varepsilon^2\Delta u - W'(u) + \varepsilon\eta_d W'(u))]. \tag{10}$$

Here $\varepsilon > 0$ and $\eta_{1,2} \in \mathbb{R}$ are parameters, $\eta_d = \eta_2 - \eta_1$, W is a double–well–potential with $W'(-1) = W'(0) = W'(u_+) = 0$, for some $u_+ > 0$, and $\mathcal{G}$ is an operator ensuring mass-conservation, e.g., $\mathcal{G}f = f - \frac{1}{|\Omega|}\int_\Omega f(x)\,\mathrm{d}x$. In suitable parameter regimes, (10) is extremely rich in pattern formation. The basic building blocks are straight and curved "channels", i.e., bilayer interfaces between $u \equiv -1$ and some positive u, which show "pearling" and "meander" instabilities, leading to more complex patterns, see Fig.2(a) for example plots.
To put (10) into `p2p2`, we set $v = \varepsilon^2\Delta u - W'(u)$ and write the stationary equation as the two component system

$$-\varepsilon^2\Delta u + W'(u) + v = 0, \quad -\varepsilon^2\Delta v + W''(u)v - \varepsilon\eta_1 v - \varepsilon\eta_d W'(u) + \varepsilon\gamma = 0, \tag{11a}$$

where γ is a Lagrange-multiplier for mass-conservation in (10). We take γ as an additional unknown, and add the equation

$$q(u) := \int_\Omega u\,\mathrm{d}x - m = 0, \tag{11b}$$

where m is a reference mass, also taken as a parameter. Thus, we now have 4 parameters $(\eta_1, \eta_2, \varepsilon, m)$, one additional unknown γ, and one additional equation, such that $n_q = 1$. We consider (10) on some rectangular domains with homogeneous Neumann BC for u and v. In this case we use both the simple and the fully assembled forms `p.fuha.sG` and `p.fuha.G`, since adaptive mesh-refinement is vital for continuation – in fact already for finding an initial solution. See the demo `fCH` for implementation details.

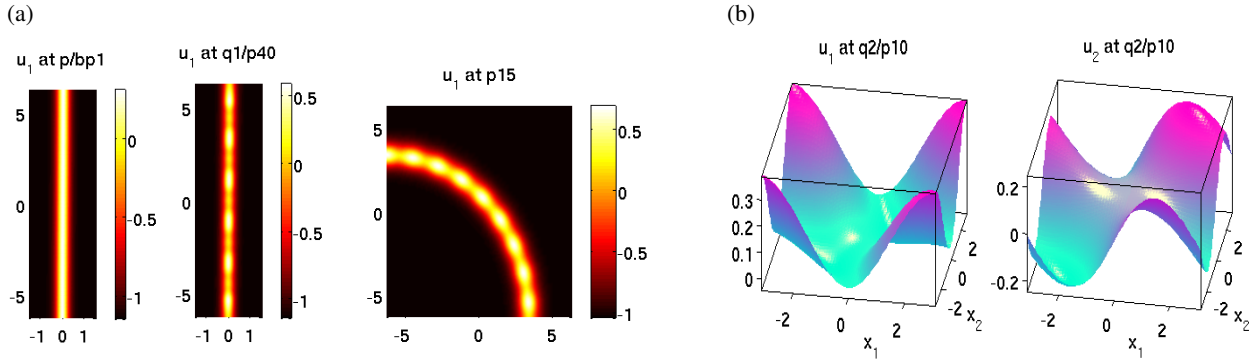

Figure 2: (a) Example plots of solutions of (11): a straight channel before and after pearling, and a curved channel after bifurcation at a pearling instability. (b) Example profiles for (13). See [3] for more details including discussion of bifurcation diagrams.

Periodic boundary conditions for rectangular shaped domains
For axis-aligned rectangular domains `p2p2` can identify opposite sides with equal grid arrangements[1] in order to generate cylindrical or toroidal geometry. The initial setup requires homogeneous Neumann boundary conditions on the sides that are to be identified, and the grid requirement is most easily realized with a mesh from `poimesh`. The boundary conditions on the remaining boundary can be arbitrary. For all calculations, the effective mesh is reduced by removing the points from one of the identified sides of the rectangle such that the solution vector `p.u` is smaller than on the initial

[1] `p2p2` only checks the boundary coordinates in the periodic direction(s) and assumes equal number of points; the transverse direction is free.

mesh. However, the full mesh and the Neumann BC are used for assembling the FEM discretized PDE and for plotting purposes.

The main function for transforming a problem with Neumann BC to periodic BC is `rec2per`, which implements periodic BC in the vertical, horizontal or both (i.e. torus) directions when the value of the argument `p.sw.bcper` is $1, 2$, or 3 respectively. In addition, the convenience function `rec2perf` can be used to load a Neumann BC solution from a file for the purpose of continuing from this solution with periodic BC. The function `rec2per` generates the matrices `p.mat.fill` and `p.mat.drop` and sets the length parameter `p.nu` to the corresponding (smaller) value. The matrix `p.mat.drop` removes the redundant entries from `p.u` via `p.mat.drop*p.u(1:p.nu)`. The matrix `p.mat.fill` extends a solution vector from the reduced to the full mesh via `p.mat.fill*p.u(1:p.nu)` by simply generating copies of entries on the periodic boundary. In particular, this 'filling' needs to be done for the fully assembled implementation of G (e.g. `acG`) and for plotting on the original Neumann grid. The justification is that in this way the FEM basis functions add up to form precisely the basis functions on the periodic domain. Internally, `p2p2`transforms the stiffness matrix K, mass matrix M and the right hand side F built with Neumann BC to corresponding matrices and vectors with periodic BC via `p.mat.fill'*`K`*p.mat.fill`, `p.mat.fill'*`M`*p.mat.fill`, and `p.mat.fill'*`F.

We refer to [3] for more details and a demo with periodic traveling waves of the Schnakenberg model on a cylinder (`schnaktravel`), which uses an additional equation to fix the translation phase and thus allow for continuation.

Example: nonlinear Bloch waves (demo `nlBloch`).

As an example for periodic boundary conditions we consider the time harmonic Gross-Pitaevskii equation

$$\omega\phi + \Delta\phi - V(x)\phi - \sigma|\phi|^2\phi = 0, \quad x \in \mathbb{R}^2 \tag{12}$$

with the periodic potential $V(x + 2\pi e_m) = V(x)$ for all $x \in \mathbb{R}^2$ and $m = 1, 2$, where e_m is the m-th Euclidean unit vector in $\mathbb{R}^2$, and $\sigma = \pm 1$. Equation (12) describes, e.g., time harmonic electromagnetic fields in nonlinear photonic crystals or Bose-Einstein condensates loaded on optical lattices. It possesses quasiperiodic solutions bifurcating from the trivial solution at spectral points $\omega_* \in \text{spec}(-\Delta + V)$, see [2, 4]. We consider here the particular case where $\omega_* = \omega_{n_*}(k_*), k_* = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$. with ω_{n_*} the n_*-th band function in the band structure $(\omega_n(k))_{n\in\mathbb{N}}$, $k \in (-1/2, 1/2]^2$. We seek a quasi-periodic nonlinear Bloch-wave ϕ of (12) with the quasiperiodicity vector k_*, i.e., $\phi(x) = e^{\mathrm{i}k_*\cdot x}\eta(x), \eta(x+2\pi e_m) = \eta(x)$ for all $x\in\mathbb{R}^2, m = 1, 2$. As shown in [4], for $\omega = \omega_* + \text{sign}(\sigma)\varepsilon^2$ with $\varepsilon > 0$ small enough such nonlinear Bloch waves exist and have the asymptotics $\phi(x) \sim \varepsilon \left(|\sigma|\|p_{n_*}(\cdot, k_*)\|^4_{L^4((-\pi,\pi)^2)}\right)^{-1/2} p_{n_*}(x, k_*)e^{\mathrm{i}k_*\cdot x}$ for $\varepsilon \to 0$. Inserting $\phi(x) = e^{\mathrm{i}k_*\cdot x}\eta(x)$ in (12) and writing it in real variables u_1, u_2, where $\eta = u_1 + \mathrm{i}u_2$, we get

$$0 = G(u_1, u_2) := -\begin{pmatrix} \Delta u_1 \\ \Delta u_2 \end{pmatrix} + 2\begin{pmatrix} k_*\cdot\nabla u_2 \\ -k_*\cdot\nabla u_1 \end{pmatrix} + (|k_*|^2 - \omega + V(x))\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} + \sigma(u_1^2 + u_2^2)\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \tag{13}$$

on the torus $\mathbb{T}^2 = \mathbb{R}^2/(2\pi\mathbb{Z}^2)$.

For a numerical example we choose $\sigma = -1$, the potential $V(x) = e^{-x_1^2}\cos(x_2),\ x \in (-\pi, \pi]^2$, and fix the bifurcation point ω_* by the choice of the second band function $n_* = 2$. We obtain numerically $\omega_* = \omega_2(1/2, 1/2) \approx 0.465$. For continuation and bifurcation in (13) we also fix $u_2 = 0$ in the lower left corner (and hence all corners) to deal with the phase–invariance of (12); see [3] for details. Figure 2(b) shows the profiles of the real and imaginary parts u_1 and u_2 respectively for a nonlinear Bloch wave at $\omega \approx 0.375$.

Discussion

We indicated some new features of `p2p2`, compared to the version documented in [7], refering to the documentation at [3] for a detailed discussion. Besides the conversion of the old demos from [7] this also includes more examples, for instance the continuation of travelling waves in the Allen–Cahn equation and in the Schnakenberg–problem. We hope that with this major upgrade a broader group of users will be reached and that it can be the basis for further research.

References

[1] A. Doelman, G. Hayrapetyan, K. Promislow, and B. Wetton. Meander and pearling of single-curvature bilayer interfaces in the functionalized Cahn-Hilliard equation. Preprint, 2012.

[2] T. Dohnal, D. Pelinovsky, and G. Schneider. Coupled-mode equations and gap solitons in a two-dimensional nonlinear elliptic problem with a separable periodic potential. *J. Nonlinear Sci.*, 19(2):95–131, 2009.

[3] T. Dohnal, J. Rademacher, H. Uecker, and D. Wetzel. p2p2 homepage: `www.staff.uni-oldenburg.de/hannes.uecker/pde2path`, 2014.

[4] T. Dohnal and H. Uecker. Bifurcation of Nonlinear Bloch waves from the spectrum in the nonlinear Gross-Pitaevskii equation. In preparation, 2014.

[5] 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.

[6] H. Uecker and D. Wetzel. The snaking width for homoclinics between spots and stripes in some Reaction–Diffusion systems. In preparation, 2014.

[7] H. Uecker, D. Wetzel, and J. Rademacher. pde2path – a Matlab package for continuation and bifurcation in 2D elliptic systems. *NMTMA (Numerical Mathematics : Theory, Methods, Applications)*, 7:58–106, 2014.