`pde2path` 3.0 – Quickstart guide and reference card

Hannes de Witt[*,1], Tomas Dohnal[2], Jens D.M. Rademacher[3], Hannes Uecker[*,4], Daniel Wetzel[*,5]

[*,1] Institut für Mathematik, Universität Oldenburg, D26111 Oldenburg, hannes.de.witt@uni-oldenburg.de,
[2] Fakultät II, Martin–Luther–Unversität Halle–Wittenberg, tomas.dohnal@mathematik.uni-halle.de
[3] Fachbereich Mathematik, Universität Bremen, D28359 Bremen, jdmr@uni-bremen.de
[4] hannes.uecker@uni-oldenburg.de, [5] daniel.wetzel@uni-oldenburg.de

Last updated: September 13, 2021, HU

Abstract

We describe version 3.0 of the PDE continuation/bifurcation package `pde2path`. After brief remarks on download and installation, we give an overview of the included demo directories, for which detailed tutorials are available on the `pde2path` homepage, and a data structure and function overview for quick reference.

Contents

1 Introduction

The `MATLAB`[1] continuation and bifurcation package `pde2path` [24, 27] treats PDE systems of type

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

where $u = u(x,t) \in \mathbb{R}^N$ (N components), $t \geq 0$, $x \in \Omega \subset \mathbb{R}^d$, $d = 1,2,3$, $M_d \in \mathbb{R}^{N\times N}$ is the (dynamical) mass matrix which may be singular, $\lambda \in \mathbb{R}^p$ is a parameter vector, and where the coefficients c, a, b and the 'nonlinearity' f may depend on x, u, λ. The boundary conditions (BCs) are of the form

$$\mathbf{n} \cdot (c\nabla u) + qu = g, \tag{2}$$

where $\mathbf{n}$ is the outer normal. Additionally there may be $n_Q \geq 1$ constraints, here written as

$$Q_j(u,\lambda) = 0, \quad j = 1,\dots,n_Q. \tag{3}$$

In the following, when we refer to (1) this always includes the BC (2) and (if applicable) the constraints (3). Often, the focus is on the steady case

$$G(u,\lambda) = 0 \text{ (and possibly } Q(u,\lambda) = 0). \tag{4}$$

The goal of `pde2path` is to be a general and easy to use (and modify and extend) toolbox to investigate bifurcations in PDEs of the (rather large) class given by (1). For detailed tutorials explaining the `pde2path` demos directories we refer to [27], and for mathematical background to [24,] and the references therein. The purpose of this document is to

- describe the basic installation of `pde2path`,
- give an overview of the `pde2path` demo directories,
- summarize the `pde2path` data structures and functions for easy reference.

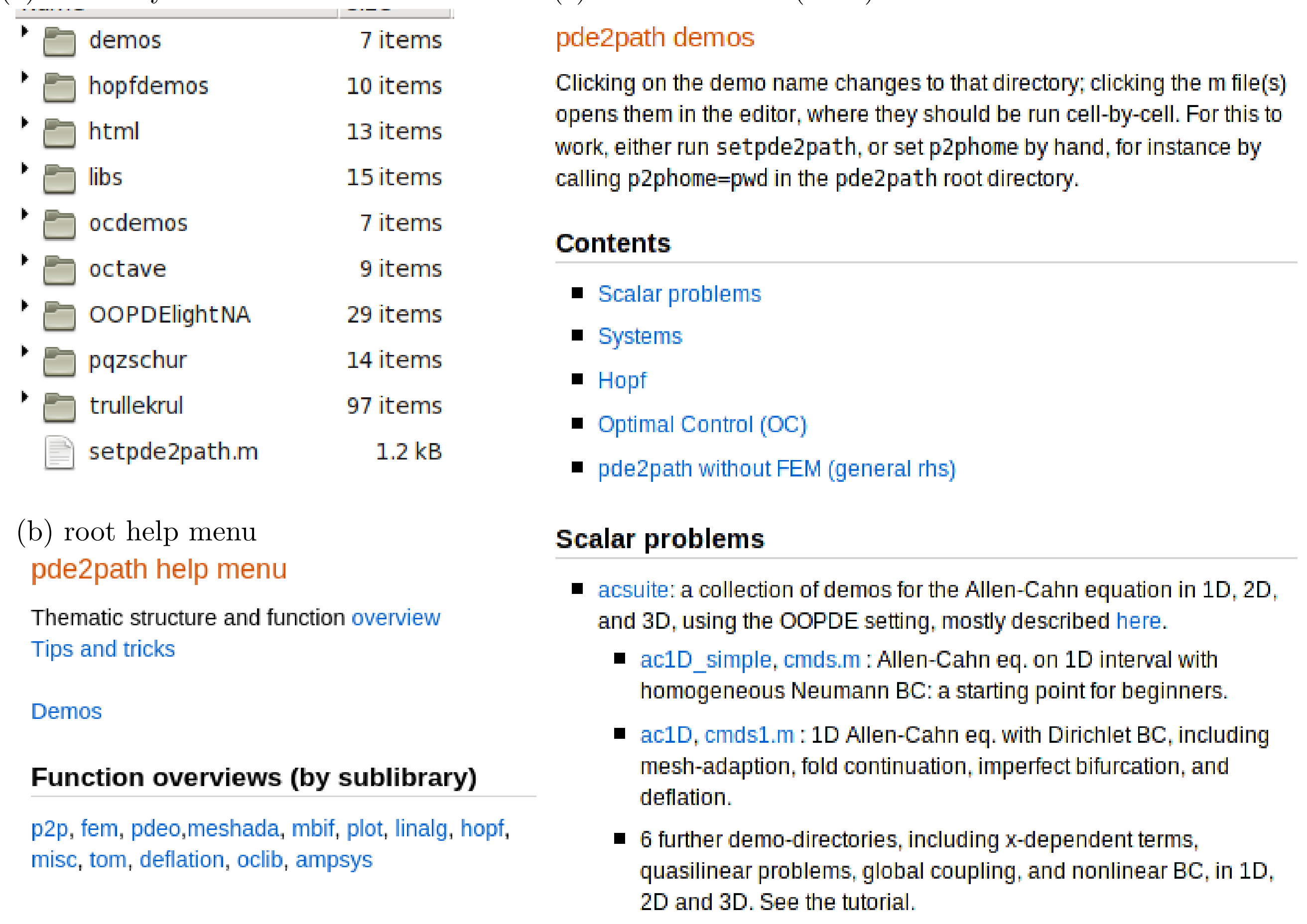
As `pde2path` evolves there will be additional features and demo directories, and there may be slight changes in file organization and data structures, which may not always be immediately updated in this guide. *Thus, for the latest documentation of* `pde2path` *we always refer to the electronic help included in the software download.* Besides `OOPDE` [16], we use four other third party softwares, namely:

- For assembling the systems for periodic orbit continuation we use (modifications of routines from) the two–point BVP package TOM [13].
- For mesh adaptation in 2D and 3D we use the package `trullekrul`.
- To compute Floquet multipliers we use `pqzschur` [12], a `MATLAB` driver for a fortran routine which computes a periodic Schur decomposition of a set of matrices.
- To solve linear systems with preconditioned iterative methods we use `ilupack` [2].

`OOPDE`, in a version with no abstract classes for compatibility with older matlab versions, (our modifications of) TOM, `trullekrul`, and `pqzschur` are included in the `pde2path` download (with permission), while `ilupack` should be downloaded at [2].[2] `OOPDE`, TOM and `trullekrul` are pure `MATLAB`, while `pqzschur` and `ilupack` require some mexing, see below. We have tested `pde2path` on a variety of standard PCs (linux, MacOS and Windows) and under various `MATLAB` versions.

[1]Most of `pde2path`'s functionality is now also available in `gnu-octave`, `www.gnu.org/software/octave/`

[2]Additionally, there are a few "public domain" (GPL) functions downloaded from the web and collected in `libs/misc`, e.g., `cheb` (from [19], used in [26]), and `BAgraphA` (modified from a function by T. Patel), `keep` (by M. Hrovat and X. Yang), and similar.



(a) Directory tree

demos	7 items
hopfdemos	10 items
html	13 items
libs	15 items
ocdemos	7 items
octave	9 items
OOPDElightNA	29 items
pqzschur	14 items
trullekrul	97 items
setpde2path.m	1.2 kB

(b) root help menu

pde2path help menu

Thematic structure and function overview
Tips and tricks

Demos

Function overviews (by sublibrary)

p2p, fem, pdeo,meshada, mbif, plot, linalg, hopf, misc, tom, deflation, oclib, ampsys

(c) demo overview (start)

pde2path demos

Clicking on the demo name changes to that directory; clicking the m file(s) opens them in the editor, where they should be run cell-by-cell. For this to work, either run setpde2path, or set p2phome by hand, for instance by calling p2phome=pwd in the pde2path root directory.

Contents

- Scalar problems
- Systems
- Hopf
- Optimal Control (OC)
- pde2path without FEM (general rhs)

Scalar problems

- acsuite: a collection of demos for the Allen-Cahn equation in 1D, 2D, and 3D, using the OOPDE setting, mostly described here.
  - ac1D_simple, cmds.m : Allen-Cahn eq. on 1D interval with homogeneous Neumann BC: a starting point for beginners.
  - ac1D, cmds1.m : 1D Allen-Cahn eq. with Dirichlet BC, including mesh-adaption, fold continuation, imperfect bifurcation, and deflation.
  - 6 further demo-directories, including x-dependent terms, quasilinear problems, global coupling, and nonlinear BC, in 1D, 2D and 3D. See the tutorial.

Figure 1: Directory tree, Root help menu, and starting part of html demo overview.

The package download `pde2path.tar.gz` (or `pde2path.zip`) unpacks to the directory `pde2path`, which contains the directory tree shown in Fig. 1(a). In this tree, `demos` and `hopfdemos` contain a number of stationary and Hopf `pde2path` demos, respectively, `html` contains help, `libs` contains the `pde2path` libraries, `ocdemos` contains the optimal control demos described in [3]. `octave` contains a `README` file explaining how to set up `pde2path` for `octave`, an `octave`–version of `OOPDE` (which features some adaptions from the `MATLAB` version), a folder `overload`, which contains modification and extensions of `pde2path` to `octave`, and a folder `octavedemos`, which shows how to adapt typical `pde2path` demos to `octave`. Further, `OOPDElightNA` is our "light" version of `OOPDE` [16], `pqzschur` contains the periodic Schur decomposition [12], which has to be mexed (see README in `pqzschur` for further instructions), and `trullekrul` [11, 10] is a powerful package for anisotropic mesh adaptation in 2D and 3D, which, besides classical error estimators is our main mesh adaptation tool.

To get started, in `MATLAB` change into the `pde2path` directory and run `setpde2path`, which also makes available the help system. Calling `p2phelp` yields the main help menu shown in Fig. 1(b). The first two topics are short thematic overviews of the data structures and main functions in `pde2path`, while clicking `p2p`, . . . , `ampsys` yields complete alphabetic function overviews of these `pde2path` libraries, with a short description of each function, which can then be clicked for further documentation.[3] Similarly, clicking on `demos` opens the demo overview in (c), with brief descriptions of and pertinent links to the demo directories and the basic script files.

[3]Help on any `pde2path` function `foo` is also given by typing `help foo` or `doc foo`, but in practice we find the alphabetic library overviews such as in Fig. 1(c) most convenient. To keep the help system functional, `clear all` should be avoided, i.e., replaced by `keep pphome`.

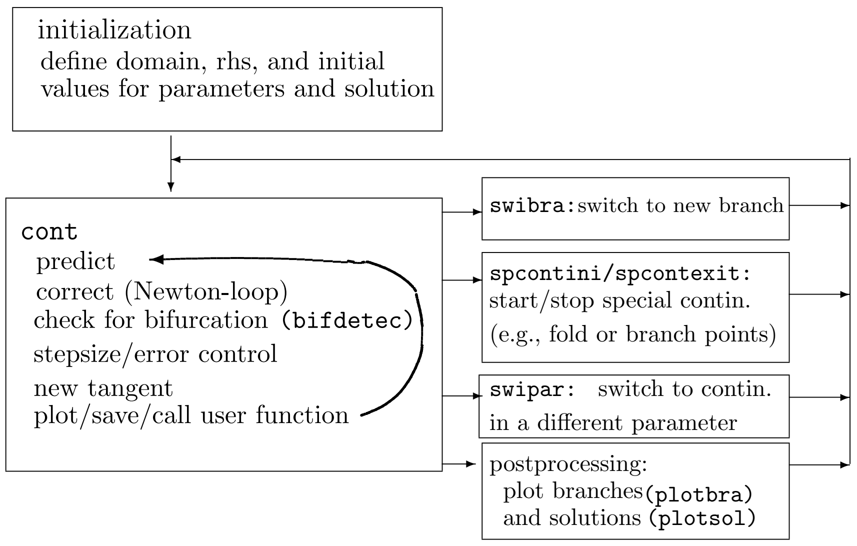

initialization
define domain, rhs, and initial values for parameters and solution
cont
predict
correct (Newton-loop)
check for bifurcation (bifdetec)
stepsize/error control
new tangent
plot/save/call user function
swibra:switch to new branch
spcontini/spcontexit:
start/stop special contin.
(e.g., fold or branch points)
swipar: switch to contin.
in a different parameter
postprocessing:
plot branches(plotbra)
and solutions (plotsol)

Figure 2: Basic flow diagram of using `pde2path`. The initialization block is typically put into a function `*init`, where `*` is the name of the problem, and it usually starts with a call of `p=stanparam(p)`, setting all `pde2path` parameters to standard values, after which the user should redefine the pertinent problem parameters. The `cont` block gives a schematic overview of the main steps in the function `cont`, where the loop is executed for a number of steps, or until some other criterion is fulfilled, for instance if a parameter leaves a predefined range. To the right there are four typical next steps after an initial (or subsequent) run of `cont` (where (a), (b) naturally assume that in some run of `cont` a branch point (BP) or fold point (FP) has been found): (a) branch switching to a bifurcating branch; (b) fold– or branch point continuation; (c) switching to continuation in a different parameter; (d) post-processing, i.e., mostly plotting. (a), (b) make sense if during `cont` a bifurcation or fold point was found. After each of these commands, `cont` can be called again to continue new branches (or further extend those already given). For convenience, all these commands are typically put into a script file `cmds.m` in "cell mode", i.e., where (groups of) commands are executed individually. This scheme basically applies to all demo directories, with some modifications, e.g.: for BPs of higher multiplicity we use `qswibra` or `cswibra` for branch switching, in the Hopf demos `swibra` and `plotsol` are replaced by the Hopf versions `hoswibra` and `hoplot`, and in some demos we use the alternative version `pmcont` instead of `cont`.

The basic flow of running a model with `pde2path` is sketched in Fig. 2. For new users, we believe that the best way to understand this scheme is to work through a number of demo problems, where `demos/acsuite` with the tutorial [18] is the easiest place to start.[4] To set up your own problem, copy the demo directory which seems closest to your problem to a new directory, then modify the functions and scripts in it. To use `ilupack`, which is useful when going to larger scale problems, mex its `MATLAB` interface, and add the `ilupack` mex directory to the `MATLAB` path.

Acknowledgment. Many thanks to Francesca Mazzia for providing TOM [13], to Uwe Prüfert for providing `OOPDE`, to Daniel Kressner for `pqzschur`, and to Kristian Ejlebjerg Jensen for `trullekrul`.

[4]We also provide testing scripts, e.g., `testp2p`, which calls `teststat, ..., testoc`, where each of these scripts calls some exemplary command files from the various demos, e.g., `teststat` starts with `testdemo('/demos/acsuite/ac1D_simple','cmds')`. Reports on these tests (success or failure) are then written to a file `log.txt` in the root directory. This is mainly for internal use, i.e., to check that everything still runs smoothly after updates, but it can also be used to check if a new installation works, and to run the demos in 'batch' mode. The data produced in the respective demo-dirs can then later be revisited for, e.g., plotting, and hence comparison with the plots in the tutorials. However, be aware that even though `testall` does not test *all* the demos, the execution may still take about 1h or more.

2 Demo overview

The following overview is intended for orientation, in particular for finding a demo similar to one's own problem, which can thus be used as a template. We group the demos into five classes: steady states for scalar PDE ($N = 1$ in (4)), steady states for PDE systems ($N > 1$ in (4)), Hopf problems, optimal control problems, and FEM–free demos. We only give brief comments, i.e., essentially:

- the equation/system studied, with hints specific features;
- hints to previous `pde2path`-manuals or newer tutorials, where applicable.

2.1 Scalar steady state and traveling wave demos

A tutorial on scalar systems is [18], see also [24, Chapter 6], dealing with (variants of) stationary Allen-Cahn (AC) problems of the form

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

with $u = u(x) \in \mathbb{R}$, $x \in \Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, Ω an interval, a rectangle or cuboid, respectively, and with various boundary conditions (BCs). The associated demo directories are in `demos/acsuite`, namely:

1. **ac1D_simple**: (5) with $x \in (-\pi, \pi)$ and Neumann BCs. Minimal example demo.
2. **ac1D**: extension of ac1D_simple. Contains slightly advanced feature such as Dirichlet BCs, mesh-adaption, fold– and branch point continuation, and restarts and deflation for imperfect bifurcations.
3. **ac1Dnlbc**: (5), 1D with nonlinear BCs
4. **ac1Dxa(b)**: a variant of (5), 1D with x-dependent coefficients, in two implementations, (a) in divergence form and (b) in non-divergence form.
5. **ac2D**: similar to ac1D, but in 2D.
6. **ac2Dwspot**: (5) with BCs that correspond to a (wandering, upon variation of some parameter) spot on the boundary. Mainly used as tutorial example for `trullekrul`, see [21].
7. **ac3D**: similar to ac1D, ac2D, but in 3D.
8. **ac3Dwspot**: similar to ac2Dwspot, but in 3D, see [21].
9. **ac2Dsq**: (5) on a square domain, hence discrete symmetry group D_4 and bifurcation points are generically double.
10. **actriangle:** (5) on an equilateral triangle (generated by `tripdeo`, hence symmetry group D_3.
11. **acdisk**: (5) on a disk domain with homogeneous Neumann BCs. Symmetry group $O(2)$, and hence many branches need phase conditions for their computation.
12. **acgc**: (5) augmented by a global coupling, i.e., $G(u) = -c\Delta u - \lambda u - u^3 + \gamma u^5 + f_{\rm gc}(u)$, where $f_{\rm gc}(u) = \delta \langle u^j \rangle u$, and $\langle v \rangle = \frac{1}{|\Omega|} \int v(x)\,{\rm d}x$ denotes a (normalized) global average. The efficient implementation relies on Sherman-Morrison formulas for linear system solvers, described in [22].
13. **acql**: a quasilinear modification of (5), i.e., of the form $-\nabla \cdot [c(u)\nabla u] - f(u) = 0$; here we treat the 1D,2D and 3D case jointly in one directory; somewhat advanced.

Periodic boundary conditions (pBCs), and in particular modifications of the `acsuite` demos to pBCs in 1D, 2D and 3D, are discussed in the tutorial [8], and

14. **ac1Dpbc, ac2Dpbc, ac3Dpbc** and **acqlpbc** under `demos/acpbc` are the associated demo directories.

The older (somewhat obsolete) `pde2path` demos for AC type models are now collected under `demos/misc`, namely `acfront` (traveling wave continuation in AC, sfem=0), and `achex` (AC on a nonstandard 2D domain, legacy setup for c, a, b, f, with x–dependent BCs, [30, §3.3]).

The next 10 demos are sub-directories of `demos/pftut`, and are described in detail in [22], see also [24, Chapters 8,10].

15. **sh**: The (quadratic-cubic) Swift–Hohenberg (SH) equation $\partial_t u = -(1+\Delta)^2 u + \lambda u + \nu u^2 - u^3$ on 1D, 2D and 3D boxes with Neumann BCs. We rewrite this scalar fourth-order equation as the two-component second order system

$$\begin{aligned} \partial_t u &= -\Delta v - 2\Delta u - (1-\lambda)u + \nu u^2 - u^3, \\ 0 &= -\partial_x^2 u + v. \end{aligned}, \quad \text{i.e.} \quad M_d = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}. \tag{6}$$

Thus we have an evolution equation for u coupled to an elliptic constraint, which in the FEM formulation can conveniently implemented using a singular mass matrix. For (6) we first compute a number of branches of periodic and localized patterns in 1D, and then in 2D and 3D, illustrating issues of multiple bifurcation points, domain and mesh symmetries, and of general rich pattern formation in higher space dimensions.

16. **shpbc**: The SH equation with periodic BCs on squares and rectangles; illustrates how to deal with the interplay of discrete and continuous symmetries, including branch–switching at BPs of higher multiplicity.
17. **shEck**: Uses branch point continuation to approximate the Eckhaus line for (6) (1D).
18. **shgc**: (6) with global coupling, i.e., $\partial_t u = -(1+\Delta)^2 u + \lambda u + \nu u^2 - u^3 - \gamma \|u\|^2 u$.
19. **CH**: The Cahn–Hilliard problem, i.e., stationary points (in particular minimizers) of the energy

$$E_\varepsilon(u) = \int_\Omega \frac{1}{2}\varepsilon^2 \|\nabla u\|^2 + W(u)\,\mathrm{d}x, \text{ under the mass constraint } \frac{1}{|\Omega|}\int_\Omega u\,\mathrm{d}x = m, \tag{7}$$

and with zero flux–BCs. Here $\Omega \subset \mathbb{R}^d$ is a bounded domain, $\varepsilon > 0$ is a parameter for the so–called interface energy, and W is a double well potential, e.g., $W(u) = -\frac{1}{2}u^2 + \frac{1}{4}u^4$. See [22, §5.2].

20. **fCH**: The functionalized Cahn–Hilliard equation from [5] describes meandering and pearling instabilities of bilayer interfaces ('channels') in a functionalized fluids. The evolution reads $\partial_t u = -\mathcal{G}[(\varepsilon^2\Delta - W''(u) + \varepsilon\eta_1)(\varepsilon^2\Delta u - W'(u)) + \varepsilon\eta_d W'(u)]$, where the operator $\mathcal{G}$ with $\mathcal{G}f = \Pi f := f - \frac{1}{|\Omega|}\int_\Omega f(x)\,\mathrm{d}x$ ensures mass conservation. Setting $v = \varepsilon^2\Delta u - W'(u)$, steady states fulfill

$$\begin{aligned} &-\varepsilon^2\Delta u + W'(u) + v = 0, \\ &-\varepsilon^2\Delta v + W''(u)v - \varepsilon\eta_1 v - \varepsilon\eta_d W'(u) + \varepsilon\gamma = 0, \end{aligned}$$

where γ is a Lagrange-multiplier for mass-conservation. We take γ as an additional unknown, and add the equation $q(u) := \int_\Omega u\,\mathrm{d}x - m = 0$, and, for numerical reasons a phase condition to keep the channels from drifting. See [22, §5.3].

21. **hexex**: The scalar equation $\Delta u + \lambda(u + u^3) = 0$ with Dirichlet BCs on a hexagonal domain as an example of higher order degenerate bifurcations, see [22, §3.3].

Problems on curved surfaces are treated in

22. **actor**: Quadratic-cubic AC equations on tori, where Δ is replaced by the Laplace–Beltrami operator (LBO) on a torus.
23. **acS**: Similar to `actor`, but on spheres S_R; branching behavior determined by spherical harmonics and O(3) symmetry.
24. **chtor**: The Cahn-Hilliard problem on a torus.

Other demo directories for scalar equations, collected under `demos/misc`, which in part have special focuses such as the linear system solvers or plotting are

25. **lss:** Demos for the linear system solvers in `pde2path`, discussed in the tutorial [29]. This also contains templates for treating larger scale problems.
26. **plotsol**: Demos for the various (solution) plot options, see [31].
27. **bratu**: $-\Delta u + 10(u - \lambda e^u) = 0$ on the unit square with zero flux BCs, originally in [30, §3.1]
28. **nlbc**: The linear equation $-\Delta u{=}0$ on the unit disk with the nonlinear BCs $\partial_n u{+}\lambda s(x,y)f(u){=}0$, $f(u) = u(1-u)$; legacy setup for c, a, b, f (sfem=0) based on the `pdetoolbox`. [6, §2.3]

2.2 System steady state and traveling wave demos

The next seven demos are sub-directories of `demos/pftut` and described in [22].

1. **schnakpat**: Steady patterns for the (modified) Schnakenberg model

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

   with diffusion matrix $D = \left(\begin{smallmatrix} 1 & 0 \\ 0 & d \end{smallmatrix}\right)$, originally in [30, §4.2].Treating (8) in 1D, 2D and 3D, with a focus on snaking branches of patterns over patterns [28], and on bifurcation points of higher multiplicity, see [22].
2. **schnakS**: (8) on spheres S_R, with Δ replaced by the pertinent LBO; branching behavior again determined by spherical harmonics and O(3) symmetry, with rather high dimensional kernels at primary bifurcation from homogeneous branch.
3. **schnaktor**: (8) on tori, again with Δ replaced by the pertinent LBO.
4. **schnakcone**: (8) on cones, with the Laplace–Beltrami operator containing mixed derivatives $\partial_x(c_{21}(x,y)\partial_y u)$ and $\partial_y(c_{12}(x,y)\partial_x u)$.
5. **accyl**: an example how to patch together two surfaces, or, more generally, two problems defined on two domains and coupled via a common boundary.
6. **cpol**: An example of bulk–surface coupling in a model for cell polarization, namely

$$\partial_t u = \varepsilon \Delta_\Gamma u + \frac{1}{\varepsilon} f(u,w), \quad x \in \Gamma, \tag{9a}$$
$$\partial_t w = \frac{1}{\varepsilon}\Delta w, \quad x \in \Omega, \tag{9b}$$
$$\partial_n w = -f(u,w), \quad x \in \Gamma, \tag{9c}$$

   where $\Omega \subset \mathbb{R}^3$ models the bulk of the cell (the cytosol), and $\Gamma = \partial\Omega$ models the cell membrane.
7. **chemtax**: The quasilinear 2-component reaction-diffusion system

$$-\begin{pmatrix} D\Delta u_1 - \lambda \nabla\cdot(u_1 \nabla u_2) \\ \Delta u_2 \end{pmatrix} - \begin{pmatrix} r u_1(1-u_1) \\ \frac{u_1}{1+u_1} - u_2 \end{pmatrix} = 0,$$

   originally in [30, §4.1], see `demos/outdated/chemtax` and also `demos/outdated/animalchem` for the old versions. Now implemented in `OOPDE`, with better Jacobians.

Some older demos concerning systems of equations are

8. **schnakfold**: Fold- and branch point continuation for (8), and comments on the various `plotbra` options; see [4].
9. **schnaktravel**: Traveling wave continuation for the 1D Schnakenberg model; an example for integral constraints and their derivatives in a system; see also [17].

10. **gp**: time–harmonic solutions of Gross–Pitaevskii equations in a rotating frame, solving real systems of the form

$$\begin{aligned}-\Delta u + (r^2-\mu)u - |U|^2u - \omega(x\partial_y v - y\partial_x v) = 0,\\ -\Delta v + (r^2-\mu)v - |U|^2v - \omega(y\partial_x u - x\partial_y u) = 0,\end{aligned}$$

where $|U|^2 = u^2+v^2$, and generalizations to more components [30, §5.1]. Inter alia a template for (2D) multi-component problems with x, y dependent advective terms.

11. **rbconv**: Rayleigh-Bénard convection in the Boussinesq approximation streamfunction form

$$\begin{aligned}-\Delta\psi + \omega \quad &= 0,\\ -\sigma\Delta\omega - \sigma R\partial_x\theta + \partial_x\psi\partial_z\omega - \partial_z\psi\partial_x\omega \quad &= 0,\\ -\Delta\theta - \partial_x\psi + \partial_x\psi\partial_z\theta - \partial_z\psi\partial_x\theta \quad &= 0,\end{aligned}$$

with various boundary conditions. Another example for advective terms, originally in [30, §5.2].

12. **twofluid**: The system

$$\begin{aligned}0 &= -\nu\Delta u_1 - (\nabla V)^\perp\cdot\nabla u_1 - (\delta+s)\partial_{x_2}u_1 + \partial_{x_2}V/L_1,\\ 0 &= -\nu\Delta u_2 - (\nabla V)^\perp\cdot\nabla u_2 - s\partial_{x_2}u_2 - \partial_{x_2}V/L_1,\\ 0 &= -\Delta V - u_1 - u_2\end{aligned}$$

over a rectangle with periodic BC in x_2 and homogeneous Dirichlet BC in x_1. See [6, §2.6.3] and [33].

13. **nlb**: Nonlinear Bloch waves fulfilling elliptic problems of the form

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

on the torus $\mathbb{T}^2 = \mathbb{R}^2/(2\pi\mathbb{Z}^2)$, cf. [7]. Thus, this is another example for periodic BCs, advection, and x-dependent coefficients. See also [6, §2.6].

The following two demos are discussed in [17] as examples for systems with continuous symmetries, requiring phase conditions (integral constraints) to remove zero eigenvalues. Thus, they are found as subdirectories of `demos/symtut`. They also explains 'freezing' to obtain traveling waves via time integration.

14. **cGL**: A complex Ginzburg–Landau equation

$$\partial_t A = \ell^2 A_{xx} + \ell s A_x + (r+\mathrm{i}\nu)A - (c_3+\mathrm{i}\mu)|A|^2A - c_5|A|^4A + \gamma, A = A(t,x)\in\mathbb{C}, \tag{10}$$

with real parameters $\ell, s, \gamma, r, \nu, c_3, \mu, c_5$ posed on the interval $x\in(-\pi,\pi)$ with periodic BCs or homogeneous Neumann BCs.

15. **FHN**: The FitzHugh-Nagumo type system in 1D

$$\begin{aligned}u_t &= \varepsilon^2 u_{xx} + su_x + u - u^3 - \varepsilon(p_3 + p_4v + p_5v^2 + p_6v^3),\\ v_t &= \varepsilon^2(v_{xx} + u - v) + sv_x,\end{aligned} \tag{11}$$

$x\in(-10,10)$, Neumann BCs, which for small $\varepsilon > 0$ has steep fronts and hence an approximate translational invariance.

2.3 Hopf demos

The mathematical background of some Hopf demos, and the numerical algorithms used, are described in [20], while [23] explains implementation details, see also [24, Chapter 7].

1. **cgl**: The complex Ginzburg–Landau equation, written as a real 2–component system
$$\partial_t \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} \Delta + r & -\nu \\ \nu & \Delta + r \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} - (u_1^2 + u_2^2) \begin{pmatrix} c_3 u_1 - \mu u_2 \\ \mu u_1 + c_3 u_2 \end{pmatrix} - c_5 (u_1^2 + u_2^2)^2 \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}. \tag{12}$$
with real parameters r, ν, c_3, μ, c_5 and $\delta > 0$, as a simple toy problem for Hopf bifurcation, over 1D, 2D and 3D cuboids with various BCs that break translational invariance.
2. **cglpbc**: (12) over 1D and 2D cuboids with periodic BCs. Hence the system O(2) symmetry (translation and reflection), and the many HBPs are at least double. Then branches of traveling waves (TWs) and standing waves (SWs) bifurcate, and we treat the TWs (time–periodic in the lab-frame) as relative equilibria, i.e., steady states in the comoving frame. Secondary bifurcations from these TWs then yield modulated TWs.
3. **cgldisk**: (12) over disks $\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 < \rho^2\}$ of radius ρ with homogeneous Neumann BCs. Again the system has O(2) symmetry, with spatial rotations taking the role of spatial translations from `cglpbc`. This leads to the bifurcation of rotating waves (RWs), which again we can continue as relative equilibria. For suitable parameters these RWs are (rotating) spirals waves, and secondary bifurcations from these yield 'meandering spirals'.
4. **cglext**: Variants of (12), including non–autonomous cases. An example for computing periodic orbits with fixed period T, and which are not generated in Hopf bifurcations.
5. **gksspirals**: The reaction diffusion system
$$\begin{aligned} \partial_t u &= d_1 \Delta u + (0.5 + r)u + v - (u^2 + v^2)(u - \alpha v), \\ \partial_t v &= d_2 \Delta v + rv - u - (u^2 + v^2)(v + \alpha u), \end{aligned}$$
on the unit disk, with Robin BCs $\partial_{\mathbf{n}} u + 10u = 0$, $\partial_{\mathbf{n}} v + 0.01v = 0$, taken from [9]. Similar to `cgldisk`, this yields the bifurcation of rotating spiral waves, and meandering spirals from secondary bifurcations.
6. **brussel**: The Brusselator system (from [32])
$$\begin{aligned} \partial_t u &= D_u \Delta u + f(u, v) - cu + dw, \\ \partial_t v &= D_v \Delta v + g(u, v), \\ \partial_t w &= D_w \Delta w + cu - dw, \end{aligned}$$
over 1D and 2D boxes with Neumann BCs. Interesting interactions of Turing branches and Turing–Hopf branches. In 2D we use preparatory steps to guess the imaginary parts of Hopf eigenvalues. We use HP and BP continuation to find pertinent bifurcation lines in the parameter plane, and also compute secondary bifurcations from periodic orbits.
7. **pollution**: The optimal control (OC) problem (see also [3])
$$V(v_0(\cdot)) := \max_{k(\cdot,\cdot)} J(v_0(\cdot), k(\cdot, \cdot)), \quad J(v_0(\cdot), k(\cdot, \cdot)) := \int_0^\infty \mathrm{e}^{-\rho t} J_{ca}(v(t), k(t)) \, \mathrm{d}t,$$
where the states v fulfill a parabolic PDE $\partial_t v = D\Delta v + g_1(v, k)$, here on an interval with homogeneous Neumann BCs. By Pontryagin's Maximum Principle we obtain the necessary first

order optimality conditions, also called canonical system,

$$\begin{aligned}\partial_t v &= D\Delta v + g_1(v,k), \quad v|_{t=0} = v_0,\\ \partial_t \lambda &= \rho\lambda + g_2(v,k) - D\Delta\lambda,\end{aligned}$$

i.e., the co-states λ fulfill a backwards diffusion equation. Therefore, here we need `pqzschur` for the computation of Floquet multipliers of time periodic orbits. This demo has been extended in `ocdemos/pollution` where we additionally explain the computation of canonical paths to canonical period states.

Hopf bifurcation with symmetries requires suitable modification of (3). Besides the demos `cglpbc`, `cgldisk` and `gksspirals` two more examples are discussed in [23], namely `mass-cons` and `kspbc2` and `kspbc4`. Two more examples with symmetries, namely `modfro` and `breathe` combine Hopf analysis with freezing and are therefore discussed in [17], and can hence be found as subdirectories of `demos/symtut`.

8. **mass-cons**: As a toy problem for mass conservation in a reaction diffusion system we consider

$$\begin{aligned}\partial_t u_1 &= \Delta u_1 + d_2\Delta u_2 + f(u_1,u_2),\\ \partial_t u_2 &= \Delta u_2 - f(u_1,u_2), \text{ in } \Omega,\end{aligned} \tag{13}$$

$f(u_1,u_2) = \alpha u_1 - u_1^3 + \beta u_1 u_2$, with parameters $d_2, \alpha, \beta \in \mathbb{R}$, and with Neumann BCs, such that the mass $m := \frac{1}{|\Omega|}\int u + v \,\mathrm{d}x$ is conserved under the evolution of (13). Spatially homogeneous steady solutions of (13) may undergo Hopf bifurcations, and the mass conservation must be appended to the pertinent Hopf equations.

9. **kspbc4** (and **kspbc2**): We consider the Kuramoto-Sivashinsky (KS) equation

$$\partial_t u = -\alpha\partial_x^4 u - \partial_x^2 u - \frac{1}{2}\partial_x(u^2), \tag{14}$$

with parameter $\alpha > 0$, on the 1D domain $x \in (-2,2)$ with periodic BCs. Here we have mass conservation and translational invariance, which both must be taken into account for the continuation of steady solutions and time periodic solutions. This is also another example for consistent set up of a 4th order equation as a 2-component 2nd order system (in `kspbc2`).

10. **modfro**: A model for autocatalysis considered in [1] is

$$\begin{aligned}\partial_t u &= a\partial_x^2 u - uf(v),\\ \partial_t v &= \partial_x^2 v + uf(v),\end{aligned} \tag{15}$$

$f(v) = v^m$ for $v \geq 0$ and 0 otherwise, where $a > 0$ and $m \geq 2.$ are parameters. We consider (15) on the domain $\Omega = (-l_x, l_x)$ with sufficiently large l_x and certain Dirichlet BCs, for which (15) has traveling fronts which may undergo Hopf bifurcations to modulated traveling fronts.

11. **breathe**: The FHN type equation

$$\begin{aligned}\partial_t u &= \partial_x^2 u + f(u,v),\\ \partial_t v &= D\partial_x^2 v + g(u,v),\end{aligned} \tag{16}$$

with homogeneous Neumann BCs, $f(u,v) = u(u-\alpha)(\beta-u) - v$, $g(u,v) = \delta(u - \gamma v)$, with $\alpha, \beta, \gamma > 0$, and $0 < \delta \ll 1$, has fronts and pulses. We consider Hopf bifurcations for standing pulses, yielding breathers, and period doubling bifurcations from the breathers [23, §6.3].

One more Hopf–demo can be found in `demos/pftut` and is described in [22, §8], namely

12. **bruosc** (and **bruosc-tpf**), dealing with the (2–component) Brusselator

$$\partial_t u_1 = a - (b+1)u_1 + u_1^2 u_2 + \alpha\cos(2\pi\omega t) + D_u\Delta u_1, \quad \partial_t u_2 = bu_1 - u_1^2 u_2 + D_v\Delta u_2, \tag{17}$$

with (optional, i.e., for $\alpha > 0$) time-periodic forcing. This is related to `hopfdemos/brussel`, but here we focus on oscillating Turing patterns arising from period–doubling bifurcations from spatially homogeneous Hopf–branches. Moreover, we explain how to implement the time periodic forcing by augmenting the 'standard implementation' with an oscillator.

2.4 OC demos

This demo directory collects a number of optimal control related problems, including an extension of `hopfdemos/pollution` explaining how to control the pollution system towards a periodic solution. However, for this somewhat special class we refer to [3] as a detailed tutorial, see also [24, Chapter 11].

2.5 FEM–free demos

In `demos/modtut` there are a number of demos which are described in [26], and which do not use the built–in FEM of `pde2path`, but are intended to give templates to implement general right hand sides G. For this, we treat the AC equation (5) on networks (aka graphs) replacing the standard Laplacian by the graph Laplacian, and on boxes via Chebychev and FFT pseudospectral methods. Additionally, we treat the Schnakenberg problem (8) on graphs and on boxes via Chebychev methods, the SH equation (6) (directly as a fourth order problem) via FFT methods, and the SH equation disks by combining Chebychev (in radius) and Fourier (in angle).

The first main modification of the standard `pde2path` setup then requires (very simpe) implementations of an alternate plotting by a function `userplot` in the respective current directory, and flagging it's use in `plotsol` by setting `p.plot.pstyle=-1`. In particular for larger scale problems, the dense Jacobians in Chebychev and FFT discretizations warrant "matrix–free methods", which are explained in `modtut/sh1Dmfree` and `modtut/sh2Dmfree`. The pros and cons of the spectral methods compared to the default FEM are further briefly discussed in [26].

2.6 Higher order FEM demos

Under `demos/hofem` we provide:

- the library `FSElib`, collecting some (partly slightly modified) functions from [15, FSELIB.14.01] which implement a quadratic (and in 1D also highr order) FEM;
- the library `hofemlib`, which contains `pde2path` interface functions to the `FSElib` (and some extensions);
- demos, essentially treating the same problems as above (Allen–Cahn, Swift–Hohenberg, Schnakenberg, over various 1D,2D and 3D domains), explaining how to run higher order FEM in `pde2path`, and its pros and cons.

For details we refer to [25].

3 Data structure overview

All data describing a given problem in `pde2path` is organized in a struct, which here and henceforth we assume is named `p` (of course any other name is also allowed). Table 1 shows the basic organization of `p`. The last four fields are supplementary in the following sense: `p.pdeo` is only needed/used if the user chooses the `OOPDE` setup (which we do in the vast majority of examples), while `p.hopf` is not needed/used for stationary problems, but initialized by `hoswibra`, i.e., by branch switching at a Hopf bifurcation point, and `p.trop` and `p.trcop` are only needed/initialized if `trullekrul` mesh adaptation is used. Most other fields are initialized via

`p=stanparam`, or, (to keep nonstandard data already present) `p=stanparam(p)`,

to 'standard values', and additional to the comments below, and for information in sync with the `pde2path` version used, we recommend to also look up fields/names in `stanparam.m`, which besides default settings also contains brief descriptions and comments. Naturally, some fields, in particular the domain geometry and mesh, and the function handles describing the PDE must be set by the user. For many problems also some of the default numerical constants such as tolerances should be adapted, and in the demos we typically do this after a first initialization with `p=stanparam`.

To understand the organization of the struct `p` we recommend to consider one of the model problems, together with one of the tutorials, where [18] is the easiest place to start, and to use the following summaries of the contents of `p` and of the `pde2path` functions as a reference card.

Table 1: Fields in the structure `p`. The distinction between `nc` and `sw` is somewhat fuzzy, as both contain variables to control the behavior of the numerics: the rule is that `nc` contains numerical constants, real or integer, while the switches in `sw` only take a finite number 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.

field	purpose	field	purpose
fuha	**fu**nction **ha**ndles, e.g., fuha.G, ...	nc	**n**umerical **c**ontrols, e.g., nc.tol, ...
sw	**sw**itches such as sw.bifcheck,...	sol	values/fields calculated at runtime
eqn	tensors c, a, b for the sfem=1 setup	mesh	mesh data (if the `pdetoolbox` is used)
plot	switches and controls for plotting	file	switches etc for file output
time	timing information	pm	pmcont switches
fsol	switches for the `fsolve` interface	nu,np	# PDE unknowns, # mesh-points
u,tau	solution and tangent	branch	branch data, see Table 24
bel	controls for lssbel (bordered elimination)	ilup	controls for lssAMG (ilupack parameters)
usrlam	vector of user set target values for the primary parameter, default usrlam=[];		
mat	various matrices and vectors, in particular the system matrices for the `sfem` $= \pm 1$ setting and other data that by default is not saved to file, including the kernel vectors and bifurcation directions in case of multiple bifurcation points.		
pdeo	OOPDE data if OOPDE is used, see §3.2	hopf	Hopf data, initialized in `hoswibra`
trop	`trullekrul` options	trcop	`trullekrul` coarsening options

3.1 Standard fields

In the following tables the default values of variables, where applicable, are those from the initialization routine `p=stanparam(p)`.

Table 2: Description of `p.fuha`. The first block pertains to `p.sw.sfem=0` (full FEM assembly of the rhs, 2D, `pdetoolbox` setting), for which G and bc are needed, and Gjac and bcjac are recommended. The second block is for `p.sw.sfem=`± 1, where sG and sGjac can be anything, but typically are composed from preassembled FEM matrices. The defaults in the third block are set by p=stanparam(p). Functions in the fourth block are only needed/recommended if p.nc.nq> 0, or for spectral continuation, respectively.

function	purpose, remarks
[c,a,f,b]=G(p,u)	compute coefficients c, a, b and f in G in the full (sfem=0) syntax
[cj,aj,bj]=Gjac(p,u)	coefficients for calculating G_u in the (sfem=0) syntax
bc=bc(p,u), bcj=bcjac(p,u)	boundary conditions, and their Jacobian
r=sG(p,u), Gu=sGjac(p,u)	residual $G(u)$ and Jacobian $G_u(u)$ in the sfem$\neq$ 0 setting using the preassembled FEM matrices such as p.mat.M, p.mat.K, ...
[p,idx]=e2rs(p,u)	elements2refine selector, used for mesh-adaptation; default is stane2rs, based on `pdejmps`.
[p,cstop]=ufu(p,brdat,ds)	user function called after each cont. step, for instance to check $\lambda_{\min} < \lambda < \lambda_{\max}$, and to give printout; cont. stops if ufu returns cstop>0; default is stanufu, which also checks if λ has passed a value in `p.usrlam`.
headfu(p)	called at start of cont, e.g. for printout; default stanheadfu
out=outfu(p,u)	generate branch data additional to bradat.m; default stanbra
savefu(p,varargin)	save solution data, default stansavefu; see also p.file for settings for saving
p=postmmod(p)	called after mesh-modification; default stanpostmeshmod
[x,p]=lss(A,b,p)	linear system solver for $Ax = b$; default is lss with $x = A\backslash u$
[x,p]=blss(B,b,p)	linear system solver for $Bx = b$, (extended or bordered linear system in arclength cont.); default is lss with $x = B\backslash u$
[x,p]=innerlss(A,b,p)	inner linear system solver, called, e.g., in `lssbel` to solve the 'bulk' system (borders removed)
q=qf(p,u), qu=qjac(p,u)	additional equation(s) $q(u)$=0, and Jac. function, see, e.g., demo `fCH`
Guuphi=spjac(p,u)	$\partial_u(\partial_u G\phi)$ for fold–or branchpoint continuation, see, e.g., demo `acfold`

Table 3: Main numerical controls in `p.nc`, with dfault values where applicable.

name & default	meaning
neq, nq	number N of equations in $G(u)$, see (4); number of additional equations (3)
tol=1e-8, imax=10	desired residual; max iterations in Newton loops
del=1e-4	stepsize for numerical differentiation
ilam	indices of active parameters; ilam(1) is the primary parameter
lammin,lammax=$\mp 1e6$	bounds for primary parameter during continuation
dsmin, dsmax	min and max arclength stepsize, current stepsize in p.sol.ds
dsinciter=imax/2	increase ds by factor dsincfac=2 if iter < dsinciter
dlammax=1	max stepsize in primary parameter
lamdtol=0.5	control to switch between arclength and natural parametrization if p.sw.para=1;
dsminbis=1e-9	min arclength in bisection for bifurcation localization
bisecmax=10	max # of bisections in bifurcation localization
nsteps=10	# of continuation steps (multiple steps for pmcont)
ntot=10000	total maximal # of continuation steps
p.nc.mu1	for bifcheck=2, start bisection if ineg changed, and $\|\text{Re}(\mu)\|$ <mu1
p.nc.mu2	for bifcheck=2, check that $\|\text{Re}(\mu)\|$ <mu2 at end of bisection
neig=[10,...]	neig(j)=# of eigenvalues to compute near the shift p.nc.eigref(j)
eigref=[0,...]	vector of shifts for eigenvalue computations, eigref(1)=0 (in general)
errbound=0	used as a trigger for mesh refinement if error>errbound> 0
amod=0	mesh-adaption each amod-th step, none if amod=0
ngen=3	number of refinement steps under mesh-refinement
bddistx=bddisty=0.1	for periodic BCs: do not refine at distance< bddistx/y from respective boundary

Table 4: Switches in `p.sw`.

name & default	meaning
bifcheck=1	0/1/2 for bif.detection off/via *LU* decomposition/via counting eigenvalues, see [20]
spcalc=1	0/1 for eigenvalue computations off/on
foldcheck=0	0/1 for fold detection off/on
jac=1	0/1 for numerical/analytical (via p.fuha.(s)jac) Jacobians for G
qjac=1	0/1 for numerical/analytical (via p.fuha.qjac) Jacobians for q
spjac=1	0/1 for numerical/analytical (via p.fuha.spjac) Jacobian for spectral point cont.
sfem=0	0/1 for full/preassembled FEM setting (-1 to flag `OOPDE` setting)
newt=0	0/1 for full/chord Newton method
bifloc=2	0 for tangent, 1 for secant, 2 for quadratic predictor in bif.localization
bcper=0	bcper> 0 indicates periodic BCs in one or more directions, see Table 5
spcont=0	0 for normal cont., 1 for bif. point cont., 2 for fold cont., 3 for Hopf point cont.
para=1	0: natural parametr.; 2: arclength; 1: automatic switching via $\dot\lambda$ <>p.nc.lamdtol. For Hopf continuation: 3: natural, 4: arclength
norm='inf'	or use any number$\geq$ 1
errcheck=0	error-estimation and handling; 0: none; 1/2: give warning/start mesh-adaption if p.sol.err>p.nc.errbound0
inter=1,verb=1	interaction and verbosity switches $\in \{0=\text{little}, 1=\text{some}, 2=\text{much}\}$
bprint=[]	indices of user-branch data for printout

Table 5: Settings for periodic boundary conditions; dir= 0 or `p.sw.pbc=0` means no periodic direction.

dim	dir	meaning	dim	dir	meaning	dim	dir	meaning
1D	1	xt	3D	1	x	3D	[1 2]	xy
2D	1	x		2	y		[1 3]	xz
	2	y		3	z		[2 3]	yz
	[1 2]	xy					[1 2 3]	xyz

Table 6: Summary of `p.mat`, which contains the FEM matrices and vectors typically generated in `setfemops` (sfem=0,1) or `oosetfemops` (sfem=-1). In the description below we assume a setup that applies to scalar equations $N = 1$. Also for systems, $N > 1$, we sometimes let M, K be matrices corresponding to one equation, i.e., let $\mathrm{M}^{-1}\mathrm{K}$ be the matrix corresponding to the "one component Neumann Laplacian", from which we compose the system stiffness and mass matrices in `sG, sGjac`. In summary, the content of `p.mat` is highly problem dependent, and must fit with `sG, sGjac`. Note that by default (i.e., in `stansavefu`), `p.mat` is *not* saved to disk with `p`.

name	meaning
M	mass matrix, used in `spcalc`, $M \in p_{n_u} \times p_{n_u}$.
K	stiffness matrix, typically used in the sfem=±1 setting
Q	boundary condition matrix to encode q in (2)
G	boundary vector for g in (2)
Kx, Ky, ...	advection matrices (if generated)
fill, drop	matrices to encode periodic boundary conditions; see [6, §2.6] and [8].
Dx, Dy, ...	matrices so encode gradients, see [18, §7].
other data	such as eigenvectors (generated/used by, e.g., `qswibra`, `cswibra`), or *LU* decompositions of Jacobians (generated/used by, e.g., `lsslu`), or preconditioners (generated/used by `lssAMG`).

Table 7: Summary of `p.mesh`. The first block only applies to the legacy setup (no `OOPDE`); in the `OOPDE` setup the pde-object `p.pdeo` contains the grid. Use the function `[po,tr,ed]=getpte(p)` to access the grid data. The second block pertains to both, the `pdetoolbox` and the `OOPDE` setup.

name	meaning
sympoi	symmetrize mesh on regular grid at startup, default=0
geo	geometry matrix in `pdetoolbox` syntax; see `pdetoolbox` documentation
p, e, t	point, edges and triangles in `pdetoolbox` syntax; see `pdetoolbox` documentation
nt, maxt	# of triangles in mesh, and max# of triangles for refinement
bp, be, bt	background–points/edges/triangles; used for coarsening before refinement in mesh adaption

Table 8: Summary of additional data in `p.sol` calculated at runtime. The current solution is stored in `p.u`, the tangent in `p.tau`, and the branch data in `p.branch`.

name	meaning	name	meaning
deta	sign of det(A)	muv	vector of eigenvalues of G_u
err	error estimate	lamd	$\dot{\lambda}$
meth	used method (nat or arc)	restart	1 to restart continuation
iter	# of iterations in last Newton loop	xi,xiq	norm weights, see [6]
ineg	# of negative eigenvalues	ds	current stepsize

Table 9: Summary of `p.file`.

name & default	meaning
count, b(f)count	counters for regular/bif./fold points; file names for regular, bif., fold points automatically composed as dir/pt`count`.mat, dir/bpt`bcount`.mat and dir/fpt`fcount`.mat
dir, pnamesw=0	directory for saving; if pnamesw=1, then set to 'name of p';
dirchecksw=0	if dirchecksw=1, then warnings given if files might be overwritten
msave=1	if msave=0, then do not save meshes with the solution data
mdir, mname	directory (default “meshes”) and file-name (generated from pname) for saving/loading meshes if p.file.msave=0;
single=0	if single=1, then save num.data in single precision (useful if low on disk-space)

Table 10: Summary of `p.plot`.

name & default	meaning	name & default	meaning
pfig=1, brfig=2	fig. nr. for sol./branch plot at runtime	ifig=6, spfig=4	info(mesh)/spectrum plot
brafig=3	fig. nr. for plotbra (a posteriori)	pcmp=1	component# for sol. plot
fs=16	fontsize	lpos=[0 0 10]	light position
cm='hot'	colormap	axis='tight'	axis type
alpha=0.1	'alpha' value for 3D plots	lev={'blue','red'}	colors for isosurfaces
lsw=1	labeling switch, (mostly) important for minimal syntax branch plotting `plotbra(p)` or `plotbraf('dir')`; see §4.6		
pstyle=1	plotstyle, options are 0,1,2,3 and 4 (in 3D, cutaway-plot), or pstyle=-1, in which case plotsol immediately call a (user–provided) function `userplot` (in the current directory).		
bpcmp=0	component# for branch plot (relative to data in outfu; last component in bradat=$\|u\|_2$ plotted if bpcmp=0), see Table 24		
udict={}	dictionary for components of u, i.e., if, e.g., u=(u_1,u_2) = (ϕ,ψ), then set `udict={\phi,\psi}`;		
auxdict={}	auxiliary variables (parameter) dictionary used for plotting, i.e., if, e.g., p.u(p.nu+1:p.nu+2)=(α,β), then set `dict={\alpha,\beta};`		
ng=20	#grid-points per direction for computing isosurfaces		

cut	for pstyle=4 and cut=$[x_1, y_1, z_1, x_2, y_2, z_2]$ only plot the part (and the triangulation) in $\Omega \cap ((x_1, x_2) \times (y_1, y_2) \times (z_1, z_2))$.

Table 11: Summary of `p.pm` (controlling `pmcont`) and `p.fsol` (controlling `fsolve`).

name & default	meaning
pm: mst=10, imax=1, resfac=0.2, runpar=0	# of parallel predictors, # of iterations in each Newton loop (adapted), factor for desired residual improvement; see [30, §4.3]. Set `runpar=0` to switch off `parfor` loops (which for instance may clash with global variables)
fsol: fsol=0, tol=1e-16, imax=5, meth, disp, opt	turn on(1)/off(0) fsol; tol and imax for fsol, and `fsolve` options. Note: `fsolve` tolerance applies to $\|G(u)\|_2^2$.

Table 12: Summary of `p.bel` and `p.ilup`, which become relevant if `lssbel, blssbel`, or `lssAMG` are chosen as linear system solvers, see §4.3 and [29].

name	meaning
bel: bw, maxit, tol ilup: maxit, droptol, maxitmax, droptolmin, droptolS	border width, max number of iterations, tolerance max # of iterations (may change), drop tolerance, upper bound for max number of GMRES iterations, minimum drop tolerance, droptolS=droptol/10 (automatically)

3.2 `OOPDE` data

`OOPDE` (object oriented PDE) is a `MATLAB`[5] FEM toolbox similar to the `MATLAB` (legacy) `pdetoolbox`, with identical interfaces in 1D, 2D, and 3D, which makes first setting up and testing a problem in 1D and then going to various domains in 2D or 3D a simple step. The object oriented (OO) setup of `OOPDE` has advantages such as tighter control of data access by the user and natural reuse resp. overload of methods by inheritance. Our basic strategy for using `OOPDE` is as follows: There are three basic templates for creating pde–objects, namely the subclasses `stanpdeo1D,stanpdeo2D, stanpdeo3D` of the `OOPDE` class `pde`. These only set up simple domains (interval, rectangle, cuboid, respectively), the grids (intervals, triangles, tetrahedra) and the finite elements (piece-wise linear continuous). Thus, calling, e.g., `p.pdeo=stanpdeo1D(lx, 2*lx/nx)`, we have `pdeo` as a pde object in `p`, i.e., the 1D domain $\Omega = (-l_x, l_x)$ with a mesh of width $2l_x/n_x$, and, by default, linear Lagrange elements associated to it.

More generally, the field `p.pdeo` of a `pde2path` struct `p` contains an object of the `OOPDE` pde class, which itself has the fields `pdeo.fem` and `pdeo.grid`. Here we give short overviews of these classes for reference. `pdeo.grid` is an object of the `OOPDE` `gridd` (super) class, which contains:

- The PTE (points–triangulation–edges) data of the grid (in `pde2path` conveniently retrieved via `[po,t,e]=getpte(p)`), and
- various (public) methods to contruct and manipulate (refine/adapt) meshes, for plotting, and, e.g., the method `makeBoundaryMatrix(bc_1,bc_2,...,bc_m)` to store boundary coefficients for assembling boundary terms, where `bc_j` contains the BCs for the j–th boundary segment. In our demos these are generated by `bc=grid.robinBC(q,g)`, where q, g are the coefficients from (2). If the BCs are the same (i.e., same q, g, which may depend on x) on all segments, then we can use `makeBoundaryMatrix(bc)`.

Many of these methods are implemented in the subclasses `grid*D`, and we in particular mention the

[5]much of `OOPDE` and hence the `OOPDE` setup of `pde2path` also run under `octave`

following:

1D Here we only have one grid constructor `grid1D.interval`, to be called as (omitting here and in the following the class identifier `grid*D`) `interval(x1,x2,hmax)`.

2D This class contains a variety of grid constructors, e.g., `square`, used in `stanpdeo1D`, with variable arguments (with obvious meanings) such as `square(xmin,xmax,ymin,ymax,hmax)` and `square(xmin,xmax,ymin,ymax,nx,ny)`, and circles and L–shapes. A general constructor is `freeGeometry(bdX)`, where `bdX` $\in \mathbb{R}^{n_b\times 2}$ contains the user chosen coordinates of the n_b boundary points, where all boundary segments are assigned 1 as identifier. Similarly, `pts2dom(X)` produces a mesh from the points in `X`, and assigns all boundary edges the identifier 1. An extension is `grid2D.freeGeoPts(bdX,X,bdi)`, where `bdX` $\in \mathbb{R}^{n_b\times 2}$ as before contains the boundary points, $X \in \mathbb{R}^{n_i\times 2}$ contains user chosen inner discretization points, and `bdi` $\in \mathbb{N}^{n_s-1}$ contains the start-indizes of the boundary segments $2,\dots,n_s$. Additionally, `grid2D` contains `ccsquare` for constructing criss–cross meshes over rectangles, and various specialized plotting methods.

3D This class again contains a number of standard shapes such as `bar`, `ccbar`, and `unitBall, cylinder`, and `ficheraCube`, and the general constructor `pts2dom`. Additionally, there is the option of using `distmesh`, [14].

From the user's point of view, most of these methods (of initial meshing) can remain hidden as long as the user is content with the selection of pde objects for which we provide convenience constructor functions such as `stanpdeo*D`, $* \in \{1,2,3\}$ or `diskpdedo`, `secpdeo`, etc.

The object `pdeo.fem` is from the the `OOPDE` (super)class `finiteElements`, which provides:

- The method `[K,M,F]=assema(grid,c,a,f)`, where `grid` is an `OOPDE` grid object, and c, a, f are the diffusion tensor, the linear term, and the nonlinearity, respectively. The outputs are the stiffness matrix K, the mass matrix M, and the right hand side F.
- The analogous method `B=convection(grid,b)`, where b is a convection tensor.
- Methods for error estimation.

Partly, these methods are defined in the subclasses `finiteElements*D`, where $* = 1, 2$ or 3, and which further split into subclasses `lagrange0*D`, `lagrange1*D`, and `bilinear3D`. These subclasses also implement

- the method `[Q,G]=assemb(grid)` for the boundary terms, where the coefficients q, g are stored via `makeBoundaryMatrix` from the `gridd` class, see above.

Again, from the user's perspective, most of this can remain hidden by choosing one of the convenience `pdeo` constructors, which in all cases choose `lagrange1*D` as the finite element class. In most of our demos, the only direct interaction with the `fem` and `grid` classes consists in calls to `fem.assema, fem.assemb` and `grid.robinBC, grid.makeBoundaryMatrix`. Examples of dealing somewhat more directly with the PTE structure of `pdeo.grid` are given in, e.g., the demos `acS, accyl`, and `cpol`.

Remark 3.1. The method `[K,M,F]=fem.assema(grid,c,a,f)` is essentially an interface to the method `fem.createMatrixEntries(grid,a,c,f)`. This is dimension *–dependent and hence implemented in subclass `finiteElements*D`, and calls `grid.aCoefficientsMpt(c,a,f)`, which is implemented in `grid*D`. The possible syntax for `c,a` and `f` is hence best checked by inspecting `grid.aCoefficientsMpt(c,a,f)` from the respective `grid*D` class, and here we mainly point out the possibilities for `c`, which naturally strongly depend on the space dimension.

In 1D, where c in $\partial_x(c(x)\partial_x u)$ is either a (scalar) constant or a scalar function, `c` as argument of `assema` can be

- the number c (the obvious and simplest case);
- a function handle `c=@cfu` (or inline function `@(x) expression`), where `c=cfu(x)` is a scalar function;

- a vector $c \in \mathbb{R}^{n_t}$ of interval midpoint values, or a vector $c \in \mathbb{R}^{n_p}$ of nodal values (which is then interpolated to interval midpoints).

The last option allows the most flexibility as here $\mathtt{c} = [\mathtt{c}_1, \dots, \mathtt{c}_\mathtt{n}]$ is prepared outside of `assema` and can for instance also depend on parameters and/or u itself.

In 2D, there are two possibilities: c can be a number (or scalar function) and is then identified with the (diagonal) tensor $\begin{pmatrix} c(x) & 0 \\ 0 & c(x) \end{pmatrix}$, or $c : \Omega \to \mathbb{R}^{2\times 2}$ (possibly constant) can be a genuine tensor. In the first case, c is passed as in 1D, namely

- If $c = \mathtt{c}$ is a number, then `K*u` simply corresponds to $-c\Delta u$;
- If $c = c(x, y)$ is a function, then c can be passed as a function handle (with signature `c=cfu(x,y)`), or a nodal or element vector, and `K` correponds to $-\partial_x(c(x, y)\partial_x u) - \partial_y(c(x, y)\partial_y u$.

If $c \in \mathbb{R}^{2\times 2}$ is a constant matrix, then the corresponding `c` can be passed in standard `MATLAB` syntax. If $c = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} : \Omega \to \mathbb{R}^{2\times 2}$ is a function, then

- c can be passed as a 2×2 cell array of function handles (with signatures `c1=cfu1(x,y), c2=cfu2(x,y), ..., c4=cfu4(x,y)`),
- or as a $2 \times (2n_p)$ or $2 \times (2n_t)$ matrix of nodal or element values, i.e.

$$\begin{aligned} \mathtt{c} = \ & [c_1(\vec{x}_1), \dots, c_1(\vec{x}_n),\ c_2(\vec{x}_1), \dots, c_2(\vec{x}_n); \\ & c_3(\vec{x}_1), \dots, c_3(\vec{x}_n,\ c_4(\vec{x}_1), \dots, c_4(\vec{x}_n)], \end{aligned} \tag{18}$$

  where $n = n_p$ or $n = n_t$. This (with $n = n_p$) is for instance the form we use in `LBcone` for the Laplace–Beltrami operator on a cone.

The 3D case works analogously; if c is a scalar, then it is identified with $c = \text{diag}(c, c, c)$, where function handles must now have the signature `c=cfu(x,y,z)`. For a matrix c=$\begin{pmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{pmatrix}$, `c` must be the corresponding 3×3 `MATLAB` matrix, or a 3×3 cell array of function handles, or analogous to (18), a $3 \times (3n_p)$ or $3 \times (3n_t)$ matrix.

Finally, similar remarks apply to assembling first order operators such as associated to (in 2D) $C = b_1\partial_x + b_2\partial_y$. See `grid*D.convCoefficients` for details. ⌋

3.3 The Hopf data

The field `p.hopf` contains the data pertaining to time-periodic orbits [20, 23]; it is typically initialized by calling `p=hoswibra(..)`. Our Hopf setup does not need any user setup additional to the functions such as `p.fuha.sG, p.fuha.sGjac` (or `p.fuha.G, p.fuha.Gjac`) already needed to describe stationary problems. The only changes of the core `p2p` library concern some queries whether we consider a Hopf problem, in which case basic routines such as `cont` call a Hopf version, i.e., `hocont`.

Table 13: Entries in `p.hopf`.

field	purpose
y	for `p.sw.para=4`: unknowns in the form $(u = (u_1, \dots, u_m) = (u(t_1), u(t_2), \dots, u(t_m))$, ($m$ time slices, y=$n_u \times m$ matrix); for `p.sw.para=3`: y augmented by $\tilde{y}$ and T, λ (($2n_u$+2) $\times$ m matrix), see [20].
y0d	for `p.sw.para=4`: $\dot{u}_0$ for the phase condition [20, (19)], ($n_u \times m$ matrix); for `p.sw.para=3`: $\dot{u}_0(0)$ for the phase condition [20, (36)], ($2n_u$+2 vector).
y0dsw	switch determining the computation of $\dot{u}_0$, default 2 (2nd order FD), see [23] for more details.

tau	tangent, see [20, (24)]
ysec	secant between two solutions (y_0, T_0, λ_0), (y_1, T_1, λ_1) for `p.sw.para=3`; $(2n_u+2)\times m$ matrix
t, T, lam	time discretization vector, current period and param.value
xi,wT	weights for the norm
x0i	index for plotting $t \mapsto u(\vec{x}($`x0i`$)$;
plot	aux. vars to control hoplot during hocont; see the description of `hoplot`; default plot=[]
wn	struct containing the winding number related settings for `initeig`
tom	struct containing TOM settings, including the mass matrix M
jac	switch to control assembly of $\partial_u \mathcal{G}$. jac=0: numerically (only recommended for testing); jac=1: via `hosjac`. Note that for `p.sw.jac=0` the local matrices $\partial_u G(u(t_j))$ are obtained via `numjac`, but this is still much faster than using `p.hopf.jac=0`.
flcheck	0 to switch off multiplier-computation during continuation, 1 to use `floq`, 2 to use `floqps`
nfloq	# of multipl. (of largest modulus) to compute (if flcheck=1)
fltol	tolerance for multiplier γ_1 (give warning if $\lvert\gamma_1 - 1\rvert >$`p.hopf.fltol`)
muv1,muv2	vectors of stable and unstable multipliers, respectively

3.4 Global variables

We generally avoid global variables and "put everything into `p`". However, if avoiding globals seems too inconvenient, we recommend to put these as subfields of `global p2pglob`, which for instance we do for Sherman–Morrison formulas for global coupling, see [22, §3.4].

4 Function overview

The `pde2path` functions are currently organized into 13 subdirectories as listed in Table 14. The main purpose of this classification is to more easily get an overview of the available functions, even though it is naturally somewhat fuzzy, e.g., `hoplot` could be assigned to the `hopf` library as well as to the `plot` library. For using `pde2path` it makes no difference in which of the subdirectories a function is, but the classification is also reflected in the `pde2path` root help menu, see Fig.1(b).

Table 14: Subdirectories of `pde2path/libs`.

subdir	contents
p2p	main steady state continuation and bifurcation (simple BPs) related routines
plot, file, misc	plotting, file-handling, and miscellaneous helper/convenience functions
pdeo, fem, linalg	pde object constructors, mesh handling and linear algebra
hopf, tom	functions related to continuation of time periodic orbits and to BVPs in time
meshada	functions for mesh adaption
deflation	functions for deflation, i.e., modified rhs to exclude known solutions
mbif	functions for bifurcation at multiple BPs
oclib	functions related to optimal control (canonical paths)

In the following we list and in part explain a number of the most important `pde2path` functions, but not all of them. In particular we omit those which are unlikely to be called directly or modified by non-expert users, and those from the optimal control library `oclib`, which pertain are documented in [3].

4.1 The stan* functions

Typically, most of the switches (`p.sw`), numerical constants (`p.nc`) and functions (`p.fuha`) that control the algorithms of `pde2path` can be set to standard values and functions via p=stanparam(p). Table 15 lists the other `stan*` functions.

Table 15: `pde2path` 'standard' settings for p.fuha, as set by p=stanparam(p); note that not all of these are always necessary, e.g., `p.fuha.postmmod` is only relevant if mesh adaption is used, and `p.fuha.innerlss` is only relevant if `p.fuha.lss=@lssbel` or `p.fuha.blss=@blssbel`.

function	used as standard setting for	purpose,remarks
out=stanbra(p,u)	p.fuha.outfu	output: out=[par;max(u_1);min(u_1)]; determines the data to be saved on p.branch. Adapt this to a given problem if other data is needed. *Note:* $\|u_1\|_{L^2}$ is already put on branch via bradat.m;
stanheadfu(p)	p.fuha.headfu	on the fly printout during continuation
[p,stop]=stanufu(p,out,ds)	p.fuha.ufu	user action (usually screen printout, but also stopping criteria), called after each continuation step
stansavefu(p)	p.fuha.savefu	save p to disk as determined via data in p.file
[p,idx]=stane2rs(p,u)	p.fuha.e2rs	idx=list of elements to refine, based on pdejmps
p=stanpostmeshmod(p)	p.fuha.postmmod	user action after mesh refinement
[x,p]=lss(A,b,p)	p.fuha.lss, p.fuha.blss, p.fuha.innerlss	linear system solver, here just an interface to \, i.e., $x = A\backslash b$; main alternatives are `lssbel` (bordered elimination) and `lssAMG` (iterative solver)
eta=stanetafu(p,np)	p.trop.etafu	standard threshold function for `trullekrul`, see [21]

4.2 Main functions for steady state problems

Table 16: Main `pde2path` functions for user calls for steady state continuation and bifurcation; some of these take auxiliary parameters, and in general the behavior is controlled by the settings in `p.nc` and `p.sw`; ... indicates additional arguments. The plotting functions are explained in §4.6, and the `OOPDE` versions of, e.g., `rec` and `stanmesh` are explained in Table 17.

function	purpose,remarks
p=stanparam(p)	sets many parameters to "standard" values; typically called during initialization; also serves as documentation of the meaning of parameters
p=cont(p), p=pmcont(p)	continuation of problem p, and parallel multi-predictor version
p=swibra(dir,bptnr,varargin)	branch–switching at simple bifurcation point dir/nr, varargin for new dir and ds
p=qswibra(dir,nr,varargin) p=cswibra(dir,nr,varargin)	branch–switching at multiple bifurcation point dir/nr via quadratic/cubic bifurcation equations, varargin for various arguments for fine tuning
plotbra(varargin)	plot branch in struct p or from file; see also p.plot for settings for plotting
plotsol(p,wnr,cmp,style)	plot solution, see also plotsolu, plotsolf, and plotEvec
p=loadp(dir,pname,varargin)	load p-data at the point pname from directory dir; varargin for new dir
p=swipar(p,var)	switch parametrization, see also swiparf
p=setlam(p,lam)	set active cont. parameter, see also getlam(p) and par=getpar(p,varargin)
geo=rec(lx,ly)	encode rectangular domain in `pdetoolbox` syntax
p=stanmesh(p,..)	generate mesh (2D, `pdetoolbox` setting)
bc=gnbc(neq,vararg)	generate `pdetoolbox`–style boundary conditions, see also the convenience functions [geo,bc]=recnbc*(lx,ly) and [geo,bc]=recdbc*(lx,ly), *=1,2

p=findbif(p,varargin)	bifurcation detection via change of stability index; alternative to bifurcation detection in cont or pmcont; can be run with larger ds, as multiple eigenvalues crossing the imaginary axis are less of a problem
p=bploc(p)	localize branch-point via extended system
p=ulamcheck(p)	check if the active continuation parameter crossed a value from p.usrlam; if yes, then compute and postprocess (plot, save) solution at that value.
p=spcontini(varargin)	initialization for "spectral continuation", in particular fold continuation
p=spcontexit(varargin)	exit spectral continuation
p=b(h)pcontini(varargin)	initialization for branch point or Hopf point continuation
p=b(h)pcontexit(varargin)	exit branch or Hopf point continuation
p=box2per(p)	transform to periodic BCs by setting p.mat.drop, p.mat.fill;
[u,...]=nloop(p,u)	Newton–loop for $(G(u), q(u)) = 0$
[u,...]=nloopext(p,u)	Newton–loop for the extended system $(G(u), q(u), p(u)) = 0$
[u,p]=deflsol(p,u1)	solution of deflated systems, set up via `deflinit`
p=meshref(p,varargin)	adaptively refine mesh, compute solution on and interpolate tangent to new mesh
p=meshadac(p,varargin)	mesh adaptation via interpolation to the (coarse) background mesh and then adaptive refinement;
p=setfn(p,name)	set output directory to name (or p, if name omitted)
screenlayout(p)	position figures for solution-plot, branch-plot and information
[Gua, Gun]=jaccheck(p)	compare Jacobian p.fuha.Gjac (resp. p.fuha.sGjac) with finite differences
[Gua, Gun]=spjaccheck(p)	compare Jacobian p.fuha.spjac with finite differences
p=setfemops(p)	generate and store FEM operators, i.e., at least the mass matrix M (if sfem=0), but also K,Q,G if sfem=1; if sfem=-1, then oosetfemops in the user directory is called.

Table 17: `OOPDE` constructor functions, creating the domain and mesh, equipped with piecewise linear Lagrange elements.

function	purpose
pde=stanpdeo1D(lx,h)	1D PDE-object constructor, i.e., interval Ω=$(-l_x, l_x)$, mesh size h
pde=stanpdeo2D(lx,ly,h)	$\Omega = (-l_x, l_x) \times (-l_y, l_y)$ with mesh size h
pde=stanpdeo2D(lx,ly,nx,ny)	$\Omega = (-l_x, l_x) \times (-l_y, l_y)$ with mesh of $n_x \times n_y$ gridpoints
pde=stanpdeo2D(lx,ly,h,sw)	analogous to stanpdeo2D(lx,ly,h), where the struct sw, containing sw.sym, can be used to prescribe the symmetry of the generated mesh. sym=0: standard, sym=1: 'pseudo criss-cross' mesh, sym=2: criss–cross
pde=stanpdeo3D(lx,ly,lz,h)	$\Omega = (-l_x, l_x) \times (-l_y, l_y) \times (-l_z, l_z)$ with mesh size h; alternatively pde=stanpdeo3D(lx,ly,lz,nx,ny,nz), pde=stanpdeo3D(lx,ly,lz,h,sw), and pde=stanpdeo3D(lx,ly,lz,nx,ny,nz,sw),
diskpdeo, freegeompdeo, ...	various further PDE Object constructor functions, see /libs/pdeo.

4.3 `linalg` and `fem`

Table 18: Main functions in `linalg` and `fem` other than already explained in, e.g., Tables 16, 17

function	purpose
[x,p]=lssbel(A,b,p)	A bordered elimination linear system solver with post-iterations; see [29], and Table 12 for controls in `p.bel`
[x,p]=blssbel(A,b,p)	increases `p.bel.bw` temporarily by 1 and calls `lssbel`
[x,p]=lssAMG(A,b,p)	ilupack linear system solver; see [29]
[x,p]=lsslu(A,b,p)	check for LU in p.LU, and use that if up to date, otherwise update

[x,p]=gclss(A,b,p)	customized lss for global coupling
[x,p]=gclsseigs(A,b,p)	`eigs` version of `gclss`
p=setbel(p,bw,...)	set bel parameters, see Table 12
p=setilup(p,dtol,maxit)	set (some) ilupack parameters, see also p=setbelilup(p,...)
[ineg,muv,V]=spcalc(Gu,p,...)	compute eigenvalues and eigenvectors of Gu
[ineg,muv,V]=vspcalc(Gu,p,...)	compute eigenvalues and eigenvectors of Gu near shifts
[po,tr,ed]=getpte(p)	get points, triangles (elements), edges from p
M=getM(p)	mass matrix
[fill,drop,nu]=getPerOp(p)	get `fill`, `drop` for periodic BCs
p=setbmesh(p)	set background mesh for mesh-adaption
g=polygong(varargin)	create polygonal domain geometry
Kx=assemadv(po,tr,b)	assemble advection matrix (`pdetoolbox` setting), see also `p.pdeo.convection(...)` (`OOPDE` setting)
Dx=makeDx(p)	make (finite difference like) 1D differentiation matrix Dx such that $\partial_x u = Dx * u$, in contrast to $\partial_x u = M^{-1}Kx * u$ using Kx. See also `[Dx,Dy]=p.pdeo.fem.gradientMatrices(p.pdeo.grid)` (2D) and `[Dx,Dy,Dz]=p.pdeo.fem.gradientMatrices(p.pdeo.grid)` (3D) (`OOPDE` setting)

4.4 Hopf

Table 19: Overview of main functions related to Hopf bifurcations and periodic orbits

name	purpose, remarks
hoswibra	branch switching at Hopf bifurcation point, see comments below
twswibra	branch switching at Hopf bifurcation point to Traveling Wave branch (which is continued as a rel.equilibrium)
poswibra	branch switching *from* Hopf orbits
hoswipar	change the active continuation parameter, see also swiparf
hoplot	plot the data contained in hopf.y. Space-time plot in 1D; in 2D and 3D: snapshots at (roughly) $t = 0$, $t = T/4$, $t = T/2$ and $t = 3T/4$; see also hoplotf;
initeig	find guess for ω_1; see also initwn
floq	compute `p.hopf.nfloq` multipliers during continuation (`p.hopf.flcheck=1`)
floqps	use periodic Schur to compute (all) multipliers during continuation (flcheck=2)
floqap, floqpsap	a posteriori versions of floq and floqps, respectively
hobra	standard–setting for p.fuha.outfu (data on branch), template for adaption to a given problem
hostanufu	standard setting for screen printout, see also hostanheadfu
plotfloq	plot previously computed multipliers
hotintxs	time integrate (1) from the data contained in p.hopf and u0, with output of $\|u(t) - u_0\|_\infty$, and saving $u(t)$ to disk at specified values
tintplot*d	plot output of hotintxs; x–t–plots for *=1, else snapshots at specified times
initwn	init vectors for computation of initial guess for spectral shifts ω_j
hogetnf	compute initial guesses for dlam, al from the normal form coefficients of bifurcating Hopf branches
hocont	main continuation routine; called by cont if p.sol.ptype>2
hostanparam	set standard parameters
hostanopt, hoMini	standard options for, and initialization of hopf.tom
hoinistep	perform 2 initial steps and compute secant, used if `p.sw.para=3`
honloopext,honloop	the arclength Newton loop, and the Newton loop with fixed λ
tomsol	use TOM to compute periodic orbit in p.sw.para=3 setting.

tomassemG	use TOM to assemble $\mathcal{G}$; see also `tomassem, tomassempbc`
gethoA	put together the extended Jacobian A for Hopf problems
hopc	the phase condition ϕ for Hopf problems, and $\partial_u\phi$
arc2tom, tom2arc	convert arclength data to tomsol data, e.g., to call tomsol for mesh adaptation. tom2arc to go back.
ulamcheckho	check for and compute solutions at user specified values in p.usrlam
hosrhs,hosrhsjac	interfaces to p.fuha.G and p.fuha.Gjac at fixed t, internal functions called by tomassempbc, together with hodummybc
horhs,hojac	similar to hosrhs, horhsjac, for p.sw.para=3, see also `hobc` and `hobcjac`

Besides `cont`, for Hopf problems the functions `initeig`, `hoswibra`, `twswibra`, `poswibra`, `hoplot`, `floqap`, `floqpsap`, `floqplot`, `hotintxs`, and `tintplot*d` are most likely to be called directly by the user, and `hobra` and `hostanufu` are likely to be adapted by the user.

4.5 Time integration

Time integration of (1) is not a key feature of `pde2path`. However, since it can be useful to obtain starting points for continuation of steady states (e.g., demo `twofluid`) , and, e.g., to study instabilities of steady states and Hopf orbits we provide a few simple time integration routines. For a given problem we essentially recommend to consider the functions in Table 20 as templates for prroblem adapted time integration.

Table 20: Templates for time integration.

function	purpose
p=tint(p,dt,nt,pmod)	time integration, semi–implicit (Euler)steps, full FEM assembling; see also `tintx` for comprehensive output of time–series.
p=tints(p,dt,nt,pmod,nffu)	time integration based on the semilinear p.sw.sfem=1 setting. If applicable, much faster than tint; again, see also `tintxs`
p=loadp2(dir,name,name0)	load u-data from name in directory dir, other p-data from name0

4.6 Plotting

Table 21 lists the main plotting routines. Since `pde2path` aims to give versatile plotting, these routines allow rather complicated argument lists. Thus, below we describe these in some detail, but otherwise refer to the demos and tutorials for examples.

Table 21: Plotting commands.

name	purpose
plotbra(varargin)	plot branch; varargin can take several forms, see below;
plotbradat(p,w,xc,yc)	plot p.branch(yc,:) vs p.branch(xc,:) to figure w
plotsol(varargin)	plot solution; varargin can take several forms, see below;
plotsolu(p,u,w,c,st)	plot component c of u in style st to figure w
faceplot(p,u)	plot u on faces of 3D domain
isoplot(p,u)	isosurface plot of u; controlled via values in p.plot
slplot(o,ng,u,fs)	slice plot (3D), o=pde-object, ng=#grid points for interp., u=sol, fs=fontsize
hoplot(p,w,c,varargin)	basic plotting routine for Hopf; varargin can take several forms; see below
xtplot(p,sol,w,c,view,title)	plot 1+1 dim soln, mainly used in Hopf and OC problems
twplot	plotting of traveling waves in lab–frame; see also rwplot, lframeplot
fouplot, plotfloq	Fourier plots, and (a posteriori) plots of Floquet multipliers

Branch plotting. Essentially, plotbra(arg1,varargin) plots the data in arg1, where `arg1` can either be a struct `p`, or a directory `dir`, in which case, `plotbra(dir,varargin)` first loads a point from disk. The behavior of plotbra is controlled by the data in p.plot and by varargin. The following calling syntaxes are typical, where dir, pt are strings that for plotbra give the directory and file name of the point to be plotted:

1. `plotbra(dir)`: convenience call; uses the branch data from the last (regular) point in `dir` scans `dir` for all available (regular, fold and bifurcation) points, and plots the branch with markers and labels on all or only some of these points, depending on `p.plot.lsw`, see Table 23. In particular, the figure-number and branch-plot component are also taken from p.plot. A useful variant thus is `plotbra(dir,'lsw',lsw)` where `lsw` overrules `p.plot.lsw`.
2. `plotbra(dir,pt,w,cmp,varargin)`: long syntax, where `pt` is the chosen point, e.g., `pt='pt5'` (5$^{\text{th}}$ computed point), or `pt='bpt2'` (2nd branch point), `w` is the figure number, `cmp` is the desired component on branch, starting with `cmp=1` for the first entry of `out=p.fuha.outfu(p,u)`, and varargin consists of 'string',value pairs according to Table 22.
   For instance, `plotbra(dir,pt,w,cmp,'cl','r','lab',[5,10],'fancy',2)` plots the branch in red and puts labels at points 5 and 10, where the 'fancy' switch controls the annotation style.
3. There are mixed forms of 1 and 2. For instance, `plotbra(dir,w,cmp,varargin)` uses the last point of dir and puts labels as in 1, but varargin can be used to overrule this, e.g., `plotbra(dir,w,cmp,'lab',10)` labels only regular point 10, and branch and fold again depending on `p.plot.lsw`.
4. A variant of `plotbra` is `plotbraf(dir,varargin)` (partly due to legacy reasons), where the first argument is always a directory. The only difference between `plotbra` and `plotbraf` is that the latter behaves like setting `lsw=31` in `plotbra`. Thus, `plotbraf(dir)` can be used to get the full information contained in directory `dir` (which typically is way too much).

Table 22: Selection of string,value combinations in varargin for fine tuning the behavior of plotbra. See [4, Tab. 4] for a full list.

name	example	meaning	name	example	meaning
fp	3	first point (on branch) to plot	ms	5	markersize (branch/hopf)
lp	7	last point to plot	fms	0	markersize (fold)
lab	[10, 13]	list of labels	lms	3	markersize (labeled)
labi	5	label each labith (regular) point	lwst	4	stable soln line width
labu	1	1 & lsw=1+x: usrlam labels	lwun	2	unstable soln line width
lsw	1	labeling switch, see Table (23)	tyst	`'-'`	stable soln line type
bplab	[2, 3]	list of branch point labels	tyun	`'--'`	unstable soln line type
fplab	1	list of fold point labels	fs	16	font size
hplab	1	list of Hopf point labels	lfs	0	label font size
fancy	0	fanciness of plotbra	cl	'r'	color

Solution plotting. Similarly to `plotbra`, there are many options how to call the main function `plotsol` for solution plotting. The easiest call is `plotsol(p)`, which plots the data contained in `p.u` with settings from `p.plot`, e.g., `p.plot.cmp` for the desired component of u. Alternatively, for instance

- `plotsol(dir)`, where `dir` is a directory, loads the last point in `dir` and proceeds as `plotsol(p)`;
- `plotsol(p,w,cmp,st)` plots component `cmp` of `p.u` to window `w`, in style `st`;
- `plotsol(dir,pt,w,cmp,st)`, loads `p` from `dir/pt` and then proceeds as `plotsol(p,w,cmp,st)`.

Additionally, there is `plotsolu(p,u,w,cmp,st)`, which plots the data from `u` instead of `p.u`. The different styles are listed in Table 25, and the behavior of plotting is further controlled by the data in

Table 23: 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

Table 24: Data on `p.branch` as generated by `bradat` (fixed) and `p.fuha.outfu=@stanbra` (customizable by user). The number `cmp` refers to the desired branch component when plotting with `plotbra`.

nr	cmp	data
1	-5	counter
2	-4	pointtype
3	-3	ineg (if `p.sw.spcalc`=1, otherwise -1)
4	-2	λ (value of active cont. param)
5	-1	err (if `p.sw.errcheck`> 0, otherwise 0)
6	0	$\|u_1\|_{L^2_a}$
$7,\dots,7+n_{aux}$	$1,\dots,n_{aux}$	auxiliary variables, typically parameters
$7+n_{aux}+1$	$n_{aux}+1$	min $|u_1|$
$7+n_{aux}+2$	$n_{aux}+2$	max $|u_1|$

`p.plot`. Nevertheless, `plotsol` is a somewhat generic routine, which sometimes needs some adaption by the user to produce publication quality results. If `p.plot.pstyle=-1`, then plotsol immediately calls `userplot`, to be user provided in the current directory. See also [31].

Table 25: Settings for pstyle.

dim	pstyle	meaning comment	dim	pstyle	meaning comment
1	1	line plot (only setting in 1D)	3	1	slice-plot
2	0	only plot the FEM mesh		2	iso-surface plot
	1	mesh-plot		3	surface-plot
	2	density-plot		4	cut-away-plot
	3	surface-plot	*	-1	to call (user provided) `userplot`

Hopf plotting. `hoplot(p,wnr,cnr,varargin)`, where `wnr` and `cnr` are the window number and component number, is the basic plotting routine for periodic orbits, contained in `p.hopf.y`. The auxiliary argument `aux=varargin` can contain a number of fields used to control its behavior. Examples are (with default values as indicated)

- `aux.lay`=[2 2]: sets the subplot-layout for the snapshots (in 2D and 3D)
- `aux.pind`=[]; set the indices, i.e., the times T*p.hopf.t(aux.pind), to be used for plotting; if `pind`=[], then the four indices 1, tl/4, tl/2, 3*tl/4 are used.
- `aux.xtics`=[]; set xtics, similar for `ytics` and `ztics`; see also `aux.cb`. (colorbar on/off)

This provides some flexibility for plotting snapshots of periodic orbits in 2D and 3D. However, often the user will adapt `hoplot` to the given problem; see [23] for examples, also dealing with movies.

4.7 Convenience functions

`pde2path` comes with a number of convenience functions, mostly collected in `libs/misc`; a brief overview is given in Table 26.

Table 26: Selected convenience functions.

name	purpose
keep(varargin)	keep varargin, clear the rest; at startup of demos used as keep pphome;
printaux(p)	print auxiliary variable index and value
printbradat(dir,pt)	print data from branch from dir/pt.mat; if pt is omitted use last pt
printdirdat(dir)	print data from points in directory dir
pcopy(olddir,newdir)	copy p2p data-directory and change file name variables, see also pmove
p2phelp	open `pde2path` help system
un=p2interp(xn,yn,u,x,y)	interpolate u from mesh x,y to un on mesh xn, yn
un=p3interp(xn,yn,zn,u,x,y,z)	interpolate u from mesh x,y,z to un on mesh xn, yn,zn

4.8 OC functions

The optimal control related functions from `libs/oclib` are a special class and are reviewed in [3].

References

[1] N. J. Balmforth, R. V. Craster, and S. J. A. Malham. Unsteady fronts in an autocatalytic system. *Proc. R. Soc. Lond., Ser. A*, 455(1984):1401–1433, 1999.

[2] M. Bollhöfer. ILUPACK V2.4, `www.icm.tu-bs.de/~bolle/ilupack/`, 2011.

[3] H. de Wit and H. Uecker. Infinite time–horizon spatially distributed optimal control problems with pde2path – algorithms and tutorial examples, arxiv:1912.11135, 2019.

[4] H. de Witt. Fold continuation in systems – a pde2path tutorial, 2017.

[5] A. Doelman, G. Hayrapetyan, K. Promislow, and B. Wetton. Meander and pearling of single-curvature bilayer interfaces in the functionalized Cahn-Hilliard equation. *SIAM J. Math. Anal.*, 46(6):3640–3677, 2014.

[6] T. Dohnal, J.D.M. Rademacher, H. Uecker, and D. Wetzel. pde2path 2.0. In H. Ecker, A. Steindl, and S. Jakubek, editors, *ENOC 2014 - Proceedings of 8th European Nonlinear Dynamics Conference, ISBN: 978-3-200-03433-4*, 2014.

[7] 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.

[8] T. Dohnal and H. Uecker. Periodic boundary conditions in pde2path, 2017.

[9] M. Golubitsky, E. Knobloch, and I. Stewart. Target patterns and spirals in planar reaction-diffusion systems. *J. Nonlinear Sci.*, 10(3):333–354, 2000.

[10] K.E. Jensen. A matlab script for solving 2d/3d miminum compliance problems using anisotropic mesh adaptation. *26th international meshing roundtable*, 203:102–114, 2017.

[11] K.E. Jensen and G. Gorman. Details of tetrahedral anisotropic mesh adaptation. *Computer Physics Communications*, 201:135–143, 2016.

[12] D. Kressner. An efficient and reliable implementation of the periodic qz algorithm. In *IFAC Workshop on Periodic Control Systems.* 2001.

[13] F. Mazzia and D. Trigiante. A hybrid mesh selection strategy based on conditioning for boundary value ODE problems. *Numerical Algorithms*, 36(2):169–187, 2004.

[14] P. Persson and G. Strang. A simple mesh generator in matlab. *SIAM Review*, 46(2):329–345, 2004.

[15] C. Pozrikidis. *Introduction to finite and spectral element methods using MATLAB®*. CRC Press, Boca Raton, FL, second edition, 2014.

[16] U. Prüfert. OOPDE, `https://tu-freiberg.de/fakult1/nmo/pruefert`, 2021.

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

[18] J.D.M. Rademacher and H. Uecker. The OOPDE setting of pde2path – a tutorial via some Allen-Cahn models, 2019.

[19] L.N. Trefethen. *Spectral methods in Matlab.* SIAM, 2002.

[20] H. Uecker. Hopf bifurcation and time periodic orbits with pde2path – algorithms and applications. *Comm. in Comp. Phys*, 25(3):812–852, 2019.

[21] H. Uecker. Using `trullekrul` in `pde2path` – anisotropic mesh–adaptation for some Allen–Cahn models in 2D and 3D, Preprint, arXiv 1912.11130 , 2019.

[22] H. Uecker. Pattern formation with pde2path – a tutorial, 2020.

[23] H. Uecker. User guide on Hopf bifurcation and time periodic orbits with pde2path, 2020.

[24] H. Uecker. *Numerical continuation and bifurcation in Nonlinear PDEs.* SIAM, 2021.

[25] H. Uecker. pde2path with higher order finite elements, 2021.

[26] H. Uecker. pde2path without finite elements, 2021. Tutorial on eqns on graphs, and spectral discretizations.

[27] H. Uecker. `www.staff.uni-oldenburg.de/hannes.uecker/pde2path`, 2021.

[28] H. Uecker and D. Wetzel. Numerical results for snaking of patterns over patterns in some 2D Selkov-Schnakenberg Reaction-Diffusion systems. *SIAM J. Appl. Dyn. Syst.*, 13(1):94–128, 2014.

[29] H. Uecker and D. Wetzel. The pde2path linear system solvers – a tutorial, 2017.

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

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

[32] L. Yang, M. Dolnik, A. M. Zhabotinsky, and I. R. Epstein. Pattern formation arising from interactions between Turing and wave instabilities. *J. Chem. Phys.*, 117(15):7259–7265, 2002.

[33] 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.