

pde2path 2.4 – Quickstart guide and reference card

Hannes de Witt^{*,1}, Tomas Dohnal², Jens D.M. Rademacher³, Hannes Uecker^{*,4}, Daniel Wetzel^{*,5}

^{*,1} Institut für Mathematik, Universität Oldenburg, D26111 Oldenburg, hannes.de.witt@uni-oldenburg.de,

² Fakultät für Mathematik, TU Dortmund, D44227 Dortmund, dohnal@mathematik.tu-dortmund.de

³ Fachbereich Mathematik, Universität Bremen, D28359 Bremen, jdmr@uni-bremen.de

⁴ hannes.uecker@uni-oldenburg.de, ⁵ daniel.wetzel@uni-oldenburg.de

April 30, 2018

Abstract

We describe the setup of version 2.4 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 now being made available on the **pde2path** homepage, and a data structure and function overview for quick reference.

Contents

1	Introduction	2
2	Demo overview	4
2.1	Scalar steady state and traveling wave demos.	5
2.2	System steady state and traveling wave demos.	6
2.3	Hopf demos.	7
2.4	OC demos	8
3	Data structure overview	9
3.1	Standard fields	9
3.2	The OOPDE setting	13
3.3	The Hopf data	13
3.4	Global variables	14
4	Function overview	14
4.1	The stan* functions	15
4.2	Main functions for steady state problems	15
4.3	linalg and fem	16
4.4	Hopf	17
4.5	Time integration	18
4.6	Plotting	18
4.7	Convenience functions	21
4.8	OC functions	21

1 Introduction

The `Matlab` continuation and bifurcation package `pde2path` treats PDE systems of type

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

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

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

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

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

In the following, when we refer to (1) this always includes the BC (2) and (if applicable) the constraints (3). The original focus of `pde2path` [UWR14, DRUW14] was on the stationary case

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

and restricted to 2D ($d = 2$), but we now treat the $d = 1, 2, 3$ dimensional cases, and with identical user interfaces for all space dimensions. This extension is based on replacing `Matlab`'s `pdetoolbox` by the FEM implementation `OOPDE` [Prü16], making `pde2path` independent of the `pdetoolbox`. Additionally, `pde2path` also includes:

- Functions and demo directories dealing with Hopf (or Poincaré–Andronov–Hopf) bifurcations and the continuation of time-periodic orbits for systems of the form (1).
- Functions and demo directories dealing with infinite time-horizon optimal control (OC) problems where the states fulfill (ill-posed) PDEs of the form (1).

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 mathematical background on the FEM discretization used to convert (1) into a system of ODEs or (for stationary problems) into an algebraic system, respectively, on continuation and bifurcation, and on the algorithms used in `pde2path`, we refer to [UWR14, DRUW14, Prü16, RU18, Uec18a, Uec18b, Uec18c]. Thus, the purpose of this document is to

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

As already said, for detailed tutorials based on the `pde2path` demos directories we refer to [Uec18d]. 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` we use three more third party softwares, namely:

- For assembling the systems for periodic orbit continuation we use (modifications of routines from) the two-point BVP package TOM [MT04].
- To compute Floquet multipliers we use `pqzschur` [Kre01], a `Matlab` driver for a fortran routine which computes a periodic Schur decomposition of a set of matrices.
- To solve linear systems with a preconditioned iterative method we use `ilupack` [Bol11].

¹In (1), $[\nabla \cdot (c \otimes \nabla u)]_i := \sum_{j=1}^N [\partial_x c_{ij11} \partial_x + \partial_x c_{ij12} \partial_y + \partial_y c_{ij21} \partial_x + \partial_y c_{ij22} \partial_y] u_j$ (i^{th} component), and similarly $[au]_i = \sum_{j=1}^N a_{ij} u_j$, $[b \otimes \nabla u]_i := \sum_{j=1}^N [b_{ij1} \partial_x + b_{ij2} \partial_y] u_j$, and $f = (f_1, \dots, f_N)$ as a column vector.

OOPDE, in a version with no abstract classes for compatibility with older matlab versions, (our modifications of) TOM, and `pqzschur` are included in the `pde2path` download (with permission), while `ilupack` should be downloaded at [Bol11]. OOPDE and TOM are pure Matlab, while `pqzschur` and `ilupack` require some mexing, see below. We have tested `pde2path` on a variety of standard PCs (with different linux distributions, MacOS and Windows) and under various Matlab versions (R2009a and later).

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 [Uec17a], `pqzschur` contains the periodic Schur decomposition [Kre01], which has to be mexed (see README in `pqzschur` for further instructions), and `OOPDElightNA` is our “light” version of OOPDE [Prü16], with No Abstract classes.

(a) Directory tree

▷	demos	20 items
▷	hopfdemos	5 items
▷	html	8 items
▷	libs	8 items
▷	ocdemos	2 items
▷	OOPDElightNA	19 items
▷	pqzschur	7 items
	setpde2path.m	489 bytes

(b) root help menu

pde2path help menu

Thematic structure and function [overview](#)
[Tips and tricks](#)

[Demos](#)

Alphabetical function overviews

[p2plib](#), [fem](#), [plot](#), [linalg](#), [hopf](#), [misc](#), [tom](#), [oc](#)

(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 \(OC\)](#)

Scalar problems

- [acfold](#), [acfold_cmds.m](#) : Allen-Cahn eq. on rectangle with homogeneous DBC and fold continuation.
- [acfront](#), [acfront_cmds.m](#) : Traveling wave continuation for Allen-Cahn eq., quasi 1D with NBC

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

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 `p2plib`, ..., `tom` yields complete alphabetic function overviews of these `pde2path` libraries, with a short description of each function, which can then be clicked for documentation.² 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.

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 the `demos/acsuite` with the tutorial [RU18] is the easiest place to start. To set up your own problem, copy the demo directory which seems closest to your problem to a new directory, then modify the

²Help 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`.

functions and scripts in it. To use `ilupack`, which is useful when going to larger scale problems, mex its `Matlab` interface, and add `ilupacks` mex directory to the `Matlab` path.

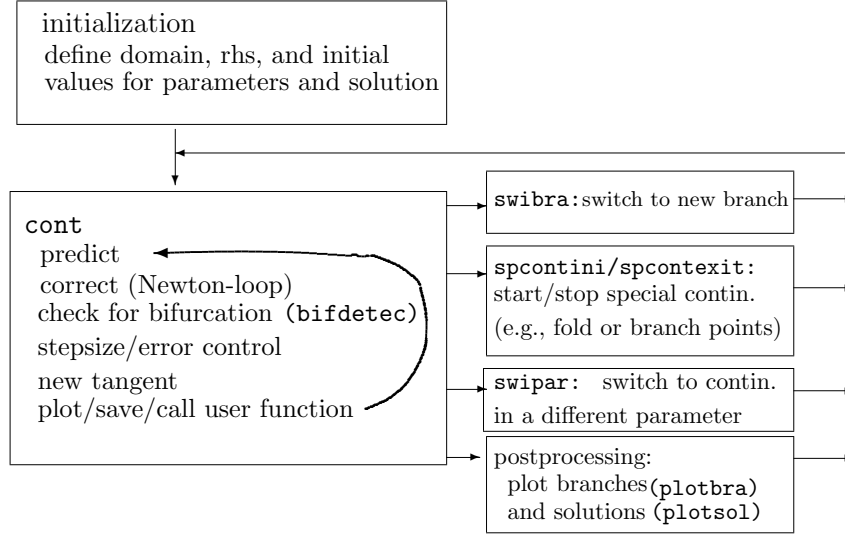


Figure 2: Basic flow diagram of using `pde2path`. The initialization block is typically put into a function `*init`, where `*` is the name of the problem, and it usually starts with a call of `p=stanparam(p)`, setting all `pde2path` parameters to standard values, after which the user should redefine 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`.

Acknowledgment. Many thanks to Francesca Mazzia for providing TOM [MT04], to Uwe Prüfert for providing OOPDE; and to Daniel Kressner for `pqzschur`.

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 four classes: steady states for scalar PDE ($N = 1$ in (4)), steady states for PDE systems ($N > 1$ in (4)), Hopf problems, and OC problems. We only give brief comments, i.e., essentially:

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

2.1 Scalar steady state and traveling wave demos.

A detailed tutorial on scalar systems is [RU18], dealing with stationary Allen-Cahn (AC) problems of the form

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

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 (BC). The associated demo directories are in **demos/acsuite**, namely:

1. **ac1D_simple**: (5) with $x \in (-\pi, \pi)$ and Neumann BC. Minimal example demo.
2. **ac1D**: extension of **ac1D_simple**. Contains slightly advanced feature such as Dirichlet BC, mesh-adaption, fold- and branch point continuation, and restarts for imperfect bifurcations.
3. **ac1Dnlbc**: (5), 1D with nonlinear BC
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. **ac3D**: similar to **ac1D**, **ac2D**, but in 3D.
7. **acgc**: (5) augmented by a global coupling, i.e., $G(u) = -c\Delta u - \lambda u - u^3 + \gamma u^5 + f_{gc}(u)$, where $f_{gc}(u) = \delta \langle u^j \rangle u$, and $\langle v \rangle = \frac{1}{|\Omega|} \int v(x) dx$ denotes a (normalized) global average. The efficient implementation relies on Sherman-Morrison formulas for linear system solvers, described in [Uec18c].
8. **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 (pBC), and in particular modifications of the **acsuite** demos to pBC in 1D, 2D and 3D, are discussed in the tutorial [DU17], and

9. **ac1Dpbc**, **ac2Dpbc**, **ac1Dpbc** 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/ac_old**, 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 BC, [UWR14, §3.3]).

Other demo directories for scalar equations, which in part have special focuses such as the linear system solvers or plotting are

10. **lss**: Demos for the linear system solvers in **pde2path**, discussed in the tutorial [UW17]. This also contains templates for treating larger scale problems.
11. **plotsol**: Demos for the various (solution) plot options, see [Wet17].
12. **bratu**: $-\Delta u + 10(u - \lambda e^u) = 0$ on the unit square with zero flux BC, originally in [UWR14, §3.1]
13. **nlbc**: The linear equation $-\Delta u = 0$ on the unit disk with the nonlinear BC $\partial_n u + \lambda s(x, y)f(u) = 0$, $f(u) = u(1 - u)$; legacy setup for c, a, b, f (sfem=0) based on the **pdetoolbox**. [DRUW14, §2.3]
14. **sh**: The (quadratic-cubic) Swift-Hohenberg equation $\partial_t u = -(1 + \Delta)^2 u + \alpha u + \nu u^3 - u^5$ on 1D, 2D and 3D boxes with Neumann BC. We rewrite this scalar fourth-order equation as the two-component second order system

$$\partial_t u = -\Delta v - 2\Delta u - (1 - \alpha)u + \nu u^3 - u^5, \quad 0 = -\partial_x^2 u + v. \quad (6)$$

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. See also demo **kspbc2** (§4.4 and [Uec17b]) for a similar construction for the Kuramoto-Sivashinsky equation. For (6) we first compute a number of stripe and snaking branches of localized patterns in 1D. Then we use (6) in 2D and 3D to illustrate the handling of multiple bifurcation points [Uec18c].

15. **gcsb**: (6) with global coupling, i.e., $\partial_t u = -(1+\Delta)^2 u + \lambda u + \nu u^2 - u^3 - \gamma \|u\|^2 u$, see [Uec18c, §3.4].
16. **hexex**: The scalar equation $\Delta u + \lambda(u + u^3) = 0$ with Dirichlet BC on a hexagonal domain as an example of higher order degenerate bifurcations, see [Uec18c, §3.3].

2.2 System steady state and traveling wave demos.

1. **schnakfold**: The (modified) Schnakenberg model

$$\partial_t U = D\Delta U + N(U, \lambda), \quad U = \begin{pmatrix} u \\ v \end{pmatrix}, \quad N(U, \lambda) = \begin{pmatrix} -u + u^2 v \\ \lambda - u^2 v \end{pmatrix} + \sigma \left(u - \frac{1}{v} \right)^2 \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad (7)$$

with, diffusion matrix $D = \begin{pmatrix} 1 & 0 \\ 0 & d \end{pmatrix}$, originally in [UWR14, §4.2]. Fold- and branch point continuation, and comments on the various `plotbra` options; see [dW17].

2. **schnakpat**: Steady patterns for (7) in 1D, 2D and 3D, with a focus on snaking branches of patterns over patterns [UW14], and on bifurcation points of higher multiplicity, see [Uec18c].
3. **schnaktravel**: Traveling wave continuation for the 1D Schnakenberg model; an example for integral constraints and their derivatives in a system; see also [RU17].
4. **chemtax**: The quasilinear 2-component reaction-diffusion system [UWR14, §4.1]

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

5. **animalchem**: chemtax on a cartoon animal skin domain [UWR14, §4.2].
6. **gp**: time-harmonic solutions of Gross–Pitaevskii equations in a rotating frame, solving real systems of the form

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

where $|U|^2 = u^2 + v^2$, and generalizations to more components [UWR14, §5.1]. Inter alia a template for (2D) multi-component problems with x, y dependent advective terms.

7. **rbconv**: Rayleigh–Bénard convection in the Boussinesq approximation streamfunction form

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

with various boundary conditions. Another example for advective terms, originally in [UWR14, §5.2].

8. **fCH**: The functionalized Cahn–Hilliard equation from [DHPW12], where 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 $q(u) := \int_{\Omega} u \, dx - m = 0$. Also an example for using `fsolve` instead of the `pde2path` Newton loop to obtain starting points for continuation. [DRUW14, §2.4].

9. **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 [DRUW14, §2.6.3] and [ZHKR15].

10. **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. [DU16]. Thus, this is another example for periodic BC, advection, and x -dependent coefficients. See also [DRUW14, §2.6].

The following two demos are discussed in [RU17] 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.

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

$$\partial_t A = \ell^2 A_{xx} + \ell s A_x + (r + i\nu)A - (c_3 + i\mu)|A|^2 A - c_5 |A|^4 A + \gamma, A = A(t, x) \in \mathbb{C}, \quad (8)$$

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

12. **FHN**: The FitzHugh–Nagumo type system in 1D

$$\begin{aligned} u_t &= \varepsilon^2 u_{xx} + s u_x + u - u^3 - \varepsilon(p_3 + p_4 v + p_5 v^2 + p_6 v^3), \\ v_t &= \varepsilon^2 (v_{xx} + u - v) + s v_x, \end{aligned} \quad (9)$$

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

2.3 Hopf demos.

The mathematical background of the first four Hopf demos, and the numerical algorithms used, are described in [Uec18a], while [Uec17b] explains implementation details. We first consider

1. **cGL**: The complex Ginzburg–Landau equation, written as a real 2-component system

$$\partial_t \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} \Delta + r & -\nu \\ \nu & \Delta + r \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} - (u_1^2 + u_2^2) \begin{pmatrix} c_3 u_1 - \mu u_2 \\ \mu u_1 + c_3 u_2 \end{pmatrix} - c_5 (u_1^2 + u_2^2)^2 \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}. \quad (10)$$

with real parameters r, ν, c_3, μ, c_5 , as a simple toy problem for Hopf bifurcation, over 1D, 2D and 3D cuboids with various BC.

2. **rot**: The reaction diffusion system

$$\begin{aligned} \partial_t u &= d_1 \Delta u + (0.5 + r)u + v - (u^2 + v^2)(u - \alpha v), \\ \partial_t v &= d_2 \Delta v + r v - u - (u^2 + v^2)(v + \alpha u), \end{aligned}$$

on the unit disk, with Robin BC $\partial_{\mathbf{n}} u + 10u = 0$, $\partial_{\mathbf{n}} v + 0.01v = 0$.

3. **brussel**: The Brusselator system (from [YDZE02])

$$\begin{aligned} \partial_t u &= D_u \Delta u + f(u, v) - cu + dw, \\ \partial_t v &= D_v \Delta v + g(u, v), \\ \partial_t w &= D_w \Delta w + cu - dw, \end{aligned}$$

over 1D and 2D boxes with homogeneous Neumann BC. This shows interesting interactions between Turing branches and Turing–Hopf branches. In 2D we use preparatory steps to guess the imaginary parts of Hopf eigenvalues.

4. **pollution**: The optimal control (OC) problem (see also [Uec17a])

$$V(v_0(\cdot)) := \max_{k(\cdot, \cdot)} J(v_0(\cdot), k(\cdot, \cdot)), \quad J(v_0(\cdot), k(\cdot, \cdot)) := \int_0^\infty e^{-\rho t} J_{ca}(v(t), k(t)) dt,$$

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 BC. 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, inter alia, here we need **pqzschur** for the computation of Floquet multipliers of time periodic orbits.

Hopf bifurcation with symmetries require suitable modifications of (3). The first two examples are discussed in [Uec17b], but the last two also combine Hopf analysis with freezing and are therefore discussed in [RU17], and can hence be found as subdirectories of **demos/symtut**.

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

$$\partial_t u_1 = \Delta u_1 + d_2 \Delta u_2 + f(u_1, u_2), \quad \partial_t u_2 = \Delta u_2 - f(u_1, u_2), \quad \text{in } \Omega, \quad (11)$$

$f(u_1, u_2) = \alpha u_1 - u_1^3 + \beta u_1 u_2$, with parameters $d_2, \alpha, \beta \in \mathbb{R}$, and with Neumann BC, such that the mass $m := \frac{1}{|\Omega|} \int u + v dx$ is conserved under the evolution of (11). Spatially homogeneous steady solutions of (11) may undergo Hopf bifurcations, and the mass conservation must be appended to the pertinent Hopf equations.

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

$$\partial_t u = -\alpha \partial_x^2 u - \partial_x^2 u - \frac{1}{2} \partial_x(u^2), \quad (12)$$

with parameter $\alpha > 0$, on the 1D domain $x \in (-2, 2)$ with periodic BC. 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**).

7. **modfro**: A model for autocatalysis considered in [BCM99] is

$$\partial_t u = a \partial_x^2 u - u f(v), \quad \partial_t v = \partial_x^2 v + u f(v), \quad (13)$$

$f(v) = v^m$ for $v \geq 0$ and 0 otherwise, where $a > 0$ and $m \geq 2$. are parameters. We consider (13) on the domain $\Omega = (-l_x, l_x)$ with sufficiently large l_x and certain Dirichlet BC, for which (13) has traveling fronts which may undergo Hopf bifurcations to modulated traveling fronts.

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

$$\partial_t u = \partial_x^2 u + f(u, v), \quad \partial_t v = D \partial_x^2 v + g(u, v), \quad (14)$$

with homogeneous Neumann BC, $f(u, v) = u(u - \alpha)(\beta - u) - v$, $g(u, v) = \delta(u - \gamma v)$, with $\alpha, \beta, \gamma > 0$, and $0 < \delta \ll 1$, has fronts and pulses. We consider Hopf bifurcations for standing pulses, yielding breathers.

2.4 OC demos

For this somewhat special class we refer to [Uec17a] as a detailed tutorial.

3 Data structure overview

Table 1 shows the basic organization of the `pde2path`-struct `p` describing a given (continuation and bifurcation) problem. The last two fields are supplementary in the following sense: `p.pdeo` is only needed/used if the user chooses the OOPDE setup, while `p.hopf` is not needed/used for stationary problems, but initialized by `hoswibra`, i.e., by branch switching at a Hopf bifurcation point.

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
<code>fuha</code>	f unction h andles, e.g., <code>fuha.G, ...</code>	<code>nc</code>	n umerical c ontrols, e.g., <code>nc.tol, ...</code>
<code>sw</code>	s witches such as <code>sw.bifcheck, ...</code>	<code>sol</code>	values/fields calculated at runtime
<code>eqn</code>	tensors c, a, b for the <code>sfem=1</code> setup	<code>mesh</code>	mesh data (if the <code>pdetoolbox</code> is used)
<code>plot</code>	switches and controls for plotting	<code>file</code>	switches etc for file output
<code>time</code>	timing information	<code>pm</code>	<code>pmcont</code> switches
<code>fsol</code>	switches for the <code>fsolve</code> interface	<code>nu,np</code>	# PDE unknowns, # mesh-points
<code>u,tau</code>	solution and tangent	<code>branch</code>	branch data, see Table 25
<code>bel</code>	controls for <code>lssbel</code> (bordered elimination)	<code>ilup</code>	controls for <code>lssAMG</code> (<code>ilupack</code> parameters)
<code>usrlam</code>	vector of user set target values for the primary parameter, default <code>usrlam=[]</code> ;		
<code>mat</code>	various matrices and vectors, in particular the system matrices for the <code>sfem = ±1</code> 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.		
<code>pdeo</code>	OOPDE data if OOPDE is used, see §3.2	<code>hopf</code>	Hopf data, initialized in <code>hoswibra</code>

To understand the organization of the struct `p` we recommend to consider one of the model problems, together with one of the tutorials, where [RU18] 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. In these summaries we first consider the legacy setup without `p.pdeo` and `p.hopf`; see §3.2 and §3.3 for the latter two.

3.1 Standard fields

In the following tables the default values of variables, where applicable, are those from the initialization routine `p=stanparam(p)`, see §4.

Table 2: Description of `p.fuha`; the first (second) block, pertains to `p.sw.sfem=0` (`p.sw.sfem=1`), respectively. In these blocks, only `G` and `bc` (`sG`) are needed if `p.sw.jac=0`, and `Gjac`, `bcjac` (`sGjac`) only if `p.sw.jac>0`. 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
<code>[c,a,f,b]=G(p,u)</code>	compute coefficients c, a, b and f in G in the full (<code>sfem=0</code>) syntax
<code>[cj,aj,bj]=Gjac(p,u)</code>	coefficients for calculating G_u in the (<code>sfem=0</code>) syntax
<code>bc=bc(p,u)</code> , <code>bcj=bcjac(p,u)</code>	boundary conditions, and their Jacobian
<code>r=sG(p,u)</code> , <code>Gu=sGjac(p,u)</code>	residual $G(u)$ and Jacobian $G_u(u)$ in the <code>sfem ≠ 0</code> setting using the preassembled FEM matrices such as <code>p.mat.M</code> , <code>p.mat.K</code> , ...
<code>[p,idx]=e2rs(p,u)</code>	<code>elements2refine</code> selector, used for mesh-adaptation; default is <code>stane2rs</code> , based on <code>pdejumps</code> .

[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=postmeshmod(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**.

name & default (where applicable)	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-8	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
dldmax=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), see also neig
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 BC: 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 [Uec18a]
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 BC in one or more directions, see Table 5
spcont=0	0 for normal cont., 1 for bif. point cont., 2 for fold 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 ≥ 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 $M^{-1}K$ 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 <code>spcalc</code> , $M \in p_{n_u} \times p_{n_u}$.
K	stiffness matrix, typically used in the <code>sfem=± 1</code> 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 [DRUW14, §2.6] and [DU17].
Dx, Dy, ...	matrices so encode gradients, see [RU18, §7].
other data	such as eigenvectors (generated/used by, e.g., <code>qswibra</code> , <code>cswibra</code>), or LU decompositions of Jacobians (generated/used by, e.g., <code>lsslu</code>), or preconditioners (generated/used by <code>lssAMG</code>).

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. Thus, generically we 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 <code>pdetoolbox</code> syntax; see <code>pdetoolbox</code> documentation
p, e, t	point, edges and triangles in <code>pdetoolbox</code> syntax; see <code>pdetoolbox</code> 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. Recall that 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 [DRUW14]
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/ptcount.mat, dir/bptbcount.mat and dir/ptfcount.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=2	plotstyle=0,1,2,3; or customize plotsol		
bpcmp=0	component# for branch plot (relative to data in outfu; last component in bradat= $\ u\ _2$ plotted if bpcmp=0), see Table 25		
udict={}	dictionary for components of u, i.e., if, e.g., $u=(u_1, u_2) = (\phi, \psi)$, then set udict ={'phi','psi'};		
auxdict={}	auxiliary variables (parameter) dictionary used for plotting, i.e., if, e.g., $p.u(p.nu+1:p.nu+2)=(\alpha, \beta)$, then set dict ={'alpha','beta'};		
ng=20	#grid-points per direction for computing isosurfaces		

Table 11: Summary of **p.pm** and **p.fsol**.

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 [UWR14, §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 [UW17].

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

3.2 The OOPDE setting

As the OOPDE setup is not yet documented in [UWR14, DRUW14], here we start with a short overview of using OOPDE in **pde2path**, see also [RU18, Uec17b, Uec17a] for extensive tutorials almost exclusively dealing with examples in the OOPDE setting. OOPDE (object oriented PDE) [Prü16] is a FEM package in **Matlab**. Inter alia, OOPDE provides, in 1D, 2D and 3D, most of the functionality that the **Matlab** **pdetoolbox** provides in 2D, and with similar basic interfaces. Thus, we use it to transfer the functionality of **pde2path** to 1D and 3D, and to also make **pde2path** independent of the **pdetoolbox** in 2D. A major difference, however, is the object oriented (OO) setup of OOPDE, which has advantages such as tighter control of data access by the user and natural reuse resp. overload of methods by inheritance, although currently we hardly use the OO aspects of OOPDE.

Our basic strategy for using OOPDE is as follows: There are three 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(1x, 2*1x/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. This is enough to, e.g., call **[K,M,dummyF]=p.pdeo.fem.assema(p.pdeo.grid,1,1,0)** to assemble the (one component) mass matrix M and stiffness matrix K (such that $M^{-1}K$ corresponds to the Neumann Laplacian), and there are useful tools to set up boundary conditions as well. After some post processing to, e.g., create the needed matrices for systems of PDEs, we typically save these system matrices in **p.mat.K**, **p.mat.M**, **p.mat.Q**, ...; see **oosetfemops** in the demo directories. After this, we can implement all functions necessary to describe the system in a standard **pde2path** way, analogous to the **pdetoolbox** **sfem=1** setting.

Table 13: Summary of **p.pdeo**.

name	meaning
fem	object of the OOPDE finiteElement class;
grid	object of the OOPDE gridd class; in particular contains the gridpoints/edges/elements which correspond to p.mesh.p , p.mesh.e , p.mesh.t in the legacy pdetoolbox setting.
time,y,...	various further (empty) entries which belong to the OOPDE pde class, but which are not used in pde2path

3.3 The Hopf data

The field **p.hopf** contains the data pertaining to time-periodic orbits [Uec18a, Uec17b]; 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 14: Entries in **p.hopf**.

field	purpose
y	for p.sw.param=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.param=3 : y augmented by \tilde{y} and T, λ ($(2n_u+2) \times m$ matrix), see [Uec18a].
y0d	for p.sw.param=4 : $M\dot{u}_0$ for the phase condition [Uec18a, (19)], ($n_u \times m$ matrix); for p.sw.param=3 : $M\dot{u}_0(0)$ for the phase condition [Uec18a, (36)], ($2n_u+2$ vector).
tau	tangent, see [Uec18a, (24)]
ysec	secant between two solutions (y_0, T_0, λ_0) , (y_1, T_1, λ_1) for p.sw.param=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)
ftol	tolerance for multiplier γ_1 (give warning if $ \gamma_1 - 1 > \mathbf{p.hopf.ftol}$)
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 [Uec18c, §3.4].

4 Function overview

The **pde2path** functions are currently organized into nine subdirectories as listed in Table 15. 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 **plot** library as well as to the **hopf** 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 15: Subdirectories of **pde2path/libs**.

subdir	contents
p2p	main steady state continuation and bifurcation (simple BPs) related routines
plot, file	plotting and file-handling
fem, linalg	mesh handling and linear algebra
hopf, tom	functions related to continuation of time periodic orbits and to BVPs in time
mbif	functions for bifurcation at multiple BPs
misc	miscellaneous helper/convenience functions
oc	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 `lib/oc`, which pertain to a rather special class of problems and are documented in [Uec17a].

4.1 The `stan*` functions

Typically, most of the switches (`p.sw`), numerical constants (`p.nc`) and functions (`p.fuha`) that determine the behavior of `pde2path` can be set to standard values and functions via `p=stanparam(p)`. Table 16 lists the other `stan*` functions.

Table 16: `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
<code>out=stanbra(p,u)</code>	<code>p.fuha.outfu</code>	output: <code>out=[par;max(u₁);min(u₁)]</code> ; determines the data to be saved on <code>p.branch</code> . Adapt this to a given problem if other data is needed. <i>Note:</i> $\ u_1\ _{L^2}$ is already put on branch via <code>bradat.m</code> ;
<code>stanheadfu(p)</code>	<code>p.fuha.headfu</code>	on the fly printout during continuation
<code>[p,stop]=stanufu(p,out,ds)</code>	<code>p.fuha.ufu</code>	user action (usually screen printout) after each continuation step
<code>stansavefu(p)</code>	<code>p.fuha.savefu</code>	save <code>p</code> to disk as determined via data in <code>p.file</code>
<code>[p,idx]=stane2rs(p,u)</code>	<code>p.fuha.e2rs</code>	<code>idx</code> =list of elements to refine, based on <code>pdejumps</code>
<code>p=stanpostmeshmod(p)</code>	<code>p.fuha.postmmod</code>	user action after mesh refinement
<code>[x,p]=lss(A,b,p)</code>	<code>p.fuha.lss</code> , <code>p.fuha.blss</code> , <code>p.fuha.innerlss</code>	linear system solver, here just an interface to <code>\</code> , i.e., $x = A \backslash b$; main alternatives are <code>lssbel</code> (bordered elimination) and <code>lssAMG</code> (iterative solver)

4.2 Main functions for steady state problems

Table 17: 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 18.

function	purpose,remarks
<code>p=stanparam(p)</code>	sets many parameters to “standard” values; typically called during initialization; also serves as documentation of the meaning of parameters
<code>p=cont(p)</code> , <code>p=pmcont(p)</code>	continuation of problem <code>p</code> , and parallel multi-predictor version
<code>p=swibra(dir,bptnr,varargin)</code>	branch-switching at simple bifurcation point <code>dir/nr</code> , <code>varargin</code> for new <code>dir</code> and <code>ds</code>
<code>p=qswibra(dir,nr,varargin)</code>	branch-switching at multiple bifurcation point <code>dir/nr</code> via quadratic/cubic
<code>p=cswibra(dir,nr,varargin)</code>	bifurcation equations, <code>varargin</code> for various arguments for fine tuning
<code>plotbra(varargin)</code>	plot branch in struct <code>p</code> or from file; see also <code>p.plot</code> for settings for plotting
<code>plotsol(p,wnr,cmp,style)</code>	plot solution, see also <code>plotsolu</code> , <code>plotsolf</code> , and <code>plotEvec</code>
<code>p=loadp(dir,pname,varargin)</code>	load <code>p</code> -data at the point <code>pname</code> from directory <code>dir</code> ; <code>varargin</code> for new <code>dir</code>
<code>p=savemesh(p)</code>	saves the current mesh to <code>p.file.mdir/pname.pt*mesh.mat</code> , and updates <code>p.file.mname</code> for reload
<code>p=swipar(p,var)</code>	switch parametrization, see also <code>swiparf</code>
<code>p=setlam(p,lam)</code>	set active cont. parameter, see also <code>getlam(p)</code> and <code>par=getpar(p,varargin)</code>

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(dir,name,npar)	initialization for "spectral continuation", e.g. fold continuation
p=spcontextit(dir,name)	exit spectral continuation
p=box2per(p)	transform to periodic BC 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 $(G(u), q(u), p(u)) = 0$
p=meshref(p,varargin)	adaptively refine mesh, compute solution on and interpolate tangent to new mesh
p=meshadac(p,varargin)	mesh adaption 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)
err=errcheck(p)	calculate error-estimate
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 18: OOPDE constructor functions, creating the domain and mesh, equipped with piecewise linear Lagrange elements.

function	purpose
pde=stanpdeo1D(lx,h)	standard 1D PDE-object constructor, i.e., interval $\Omega=(-l_x, l_x)$, 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=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)	

4.3 linalg and fem

Table 19: Main functions in **linalg** and **fem** other than already explained in, e.g., Tables 17, 18

function	purpose
[x,p]=lssbel(A,b,p)	A bordered elimination linear system solver with post-iterations; see [UW17], 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 [UW17]
[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]=gclseigs(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 BC
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 20: Overview of main functions related to Hopf bifurcations and periodic orbits

name	purpose, remarks
hoswibra	branch switching at Hopf bifurcation point, see comments below
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`, the functions `initeig`, `hoswibra`, `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 listed in Table 21.

Table 21: Templates for time integration.

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

4.6 Plotting

Table 22 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 detailed examples.

Table 22: Plotting commands.

name	purpose
<code>plotbra(varargin)</code>	plot branch; varargin can take several forms, see below;
<code>plotbradat(p,w,xc,yc)</code>	plot <code>p.branch(yc,:)</code> vs <code>p.branch(xc,:)</code> to figure w
<code>plotsol(varargin)</code>	plot solution; varargin can take several forms, see below;
<code>plotsolu(p,u,w,c,st)</code>	plot component c of u in style st to figure w
<code>faceplot(p,u)</code>	plot u on faces of 3D domain
<code>isoplot(p,u)</code>	isosurface plot of u; controlled via values in <code>p.plot</code>
<code>slplot(o,ng,u,fs)</code>	slice plot (3D), <code>o=pde-object</code> , <code>ng=#grid points for interp.</code> , <code>u=sol</code> , <code>fs=fontsize</code>
<code>hoplot(p,w,c,varargin)</code>	basic plotting routine for Hopf orbits; varargin can take several forms; see below
<code>xtplot(p,sol,w,c,view,tit)</code>	plot 1+1 dim soln, mainly used in OC problems

Branch plotting. Essentially, `plotbra(arg1,varargin)` plots the data in `arg1`, where `arg1` can either be a main 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 24. 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'` (5th computed point), or `pt='bpt2'` (2nd branch point), `w` is the figure number, `cmp` is the de-

sired 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 23.

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 23: Selection of string,value combinations in `varargin` for fine tuning the behavior of `plotbra`. See [dW17, 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 labi^{th} (regular) point	lwst	4	stable soln line width
labu	1	1 & <code>lsw=1+x</code> : force usrlam labels	lwun	2	unstable soln line width
lsw	1	labeling switch, see Table (24)	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 <code>plotbra</code>	cl	'r'	color

Table 24: 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 25: 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 <code>p.sw.spcalc=1</code> , otherwise -1)
4	-2	λ (value of active cont. param)
5	-1	err (if <code>p.sw.errcheck > 0</code> , otherwise 0)
6	0	$\ u_1\ _{L^2_\lambda}$
$7, \dots, 7 + n_{aux}$	$1, \dots, n_{aux}$	auxiliary variables, typically parameters
$7 + n_{aux} + 1$	$n_{aux} + 1$	$\min u_1 $
$7 + n_{aux} + 2$	$n_{aux} + 2$	$\max u_1 $

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 26, and the behavior of plotting is further controlled by the data in `p.plot`. Nevertheless, `plotsol` is a somewhat generic routine, which sometimes needs some adaption by the user to produce publication quality results. See also [Wet17].

Table 26: Settings for `pstyle`.

dim	pstyle	meaning comment	dim	pstyle	meaning comment
1	1	line plot (only setting in 1D)	2	3	surface-plot
2	0	only plot the FEM mesh	3	1	slice-plot
	1	mesh-plot		2	iso-surface plot
	2	density-plot		3	surface-plot

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 $T \cdot p.hopf.t(\text{aux.pind})$, to be used for plotting; if `pind=[]`, then the four indices 1, $t/4$, $t/2$, $3 \cdot t/4$ 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 [Uec17b] 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 27.

Table 27: Selected convenience functions.

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

4.8 OC functions

The optimal control related functions from `libs/oc` are a special class and are reviewed in [Uec17a].

References

- [BCM99] N. J. Balmforth, R. V. Craster, and S. J. A. Malham. Unsteady fronts in an autocatalytic system. *R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci.*, 455(1984):1401–1433, 1999.
- [Bol11] M. Bollhöfer. ILUPACK V2.4, www.icm.tu-bs.de/~bolle/ilupack/, 2011.
- [DHPW12] 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.
- [DRUW14] T. Dohnal, J. 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.
- [DU16] T. Dohnal and H. Uecker. Bifurcation of Nonlinear Bloch waves from the spectrum in the nonlinear Gross-Pitaevskii equation. *J. Nonlinear Sci.*, 26(3):581–618, 2016.
- [DU17] T. Dohnal and H. Uecker. Periodic boundary conditions in pde2path, 2017.
- [dW17] H. de Witt. Fold continuation in systems – a pde2path tutorial, 2017.
- [Kre01] D. Kressner. An efficient and reliable implementation of the periodic qz algorithm. In *IFAC Workshop on Periodic Control Systems*. 2001.
- [MT04] 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.
- [Prü16] U. Prüfert. OOPDE, www.mathe.tu-freiberg.de/nmo/mitarbeiter/uwe-pruefert/software, 2016.
- [RU17] J. Rademacher and H. Uecker. Symmetries, freezing, and Hopf bifurcations of modulated traveling waves in pde2path, 2017.
- [RU18] J. Rademacher and H. Uecker. The OOPDE setting of pde2path – a tutorial via some Allen-Cahn models, 2018.
- [Uec17a] H. Uecker. Infinite time-horizon spatially distributed optimal control problems with pde2path – a tutorial, 2017.
- [Uec17b] H. Uecker. User guide on Hopf bifurcation and time periodic orbits with pde2path, 2017.
- [Uec18a] H. Uecker. Hopf bifurcation and time periodic orbits with pde2path – algorithms and applications, *Comm. in Comp. Phys.*, to appear, 2018.
- [Uec18b] H. Uecker. Multiple bifurcation points in pde2path, Preprint, 2018.
- [Uec18c] H. Uecker. Pattern formation with pde2path – a tutorial, 2018.
- [Uec18d] H. Uecker. www.staff.uni-oldenburg.de/hannes.uecker/pde2path, 2018.
- [UW14] H. Uecker and D. Wetzel. Numerical results for snaking of patterns over patterns in some 2D Selkov-Schnakenberg Reaction-Diffusion systems. *SIADS*, 13-1:94–128, 2014.
- [UW17] H. Uecker and D. Wetzel. The pde2path linear system solvers – a tutorial, 2017.
- [UWR14] H. Uecker, D. Wetzel, and J. Rademacher. pde2path – a Matlab package for continuation and bifurcation in 2D elliptic systems. *NMTMA*, 7:58–106, 2014.
- [Wet17] D. Wetzel. A pde2path `plotsol` tutorial, 2017.
- [YDZE02] 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.
- [ZHKR15] D. Zhelyazov, D. Han-Kwan, and J. D. M. Rademacher. Global stability and local bifurcations in a two-fluid model for tokamak plasma. *SIAM J. Appl. Dyn. Syst.*, 14(2):730–763, 2015.