

# User guide on Hopf bifurcation and time periodic orbits with `pde2path`

Hannes Uecker

Institut für Mathematik, Universität Oldenburg, D26111 Oldenburg, hannes.uecker@uni-oldenburg.de

December 22, 2017

## Abstract

We describe the setup for using the `pde2path` libraries for Hopf bifurcation and continuation of branches of periodic orbits, and implementation details of the associated demo directories. See [Uec17a] for a description of the basic algorithms and the mathematical background of the examples. Additionally we explain the treatment of Hopf bifurcations in systems with continuous symmetries, e.g., of Hopf bifurcation from traveling waves to modulated traveling waves.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>The cGL equation as an introductory example: Demo cGL</b>	<b>4</b>
2.1	1D . . . . .	6
2.2	Remarks on Floquet multipliers and time integration . . . . .	9
2.3	2D . . . . .	11
2.4	3D . . . . .	11
<b>3</b>	<b>Two reaction diffusion systems</b>	<b>12</b>
3.1	Rotating patterns on a disk: Demo <code>rot</code> . . . . .	12
3.2	An extended Brusselator: Demo <code>brussel</code> . . . . .	15
<b>4</b>	<b>A canonical system from optimal control: Demo pollution</b>	<b>17</b>
<b>5</b>	<b>Hopf bifurcation with symmetries</b>	<b>19</b>
5.1	Mass conservation: Demo <code>mass-cons</code> . . . . .	20
5.2	Mass and phase constraints: Demos <code>kspbc4</code> and <code>kspbc2</code> . . . . .	23
<b>A</b>	<b>A collection of formulas</b>	<b>26</b>
<b>B</b>	<b>Data structure and function overview</b>	<b>30</b>

## 1 Introduction

In [Uec17a] we describe the algorithms in `pde2path` to study Hopf bifurcations in PDEs of the form

$$\partial_t u = -G(u, \lambda), \quad u = u(x, t), \quad x \in \Omega, \quad t \in \mathbb{R}, \quad (1.1)$$

where  $G(u, \lambda) := -\nabla \cdot (c \otimes \nabla u) + au - b \otimes \nabla u - f$ . Here  $u = u(x, t) \in \mathbb{R}^N$ ,  $x \in \Omega$  with  $\Omega \subset \mathbb{R}^d$  some bounded domain,  $d = 1, 2, 3$ ,  $\lambda \in \mathbb{R}^p$  is a parameter (vector), and the diffusion, advection and

linear tensors  $c, b, a$ , and the nonlinearity  $f$ , can depend on  $x, u, \nabla u$ , and parameters. The boundary conditions (BC) for (1.1) are of “generalized Neumann” form, i.e.,

$$\mathbf{n} \cdot (c \otimes \nabla u) + qu = g, \quad (1.2)$$

where  $\mathbf{n}$  is the outer normal and again  $q \in \mathbb{R}^{N \times N}$  and  $g \in \mathbb{R}^N$  may depend on  $x, u, \nabla u$  and parameters, and over rectangular domains there additionally is the possibility of periodic BC in one or more directions. See [Uec17a], and the steady state tutorials at [Uec17c], for more details on  $c, b, \dots, g$ .

`pde2path` spatially discretizes the PDE (1.1), (1.2) via piecewise linear finite elements, leading to the high-dimensional ODE problem (with a slight misuse of notation)

$$M\dot{u} = -G(u, \lambda), \quad u = u(t) \in \mathbb{R}^{n_u}, \quad G(u) = Ku - Mf(u), \quad (1.3)$$

where  $n_u = n_p N$  is the number of unknowns, with  $n_p$  the number of grid points. In (1.3),  $M$  is the mass matrix,  $K$  is the stiffness matrix, which typically corresponds to the diffusion term  $-\nabla \cdot (c \otimes \nabla u)$ , and  $Mf : \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_u}$  contains the rest, which we often also call the ‘nonlinearity’. However, (1.3) is really a sort of symbolic notation to express the spatially discretized version of (1.1), and  $K$  in (1.3) can also involve nonlinear terms and first order derivatives coming from  $b \otimes \nabla u$  in (1.1). See, e.g., [RU17b] for more details.

Here we first present implementation details for the four Hopf bifurcation test problems considered in [Uec17a], thus giving a tutorial on how to treat Hopf bifurcations in `pde2path`. See [Uec17c] for download of the package, including the demo directories, and various documentation and tutorials. In particular, three of the implementations here use a very similar `OOPDE` [Prü16] setup like those in [RU17b], and since the Hopf examples are somewhat more involved than the steady case we recommend to new users of `pde2path` to first look into [RU17b], which starts with some simple steady state problems.

The first Hopf demo problem is a cubic–quintic complex Ginzburg–Landau (cGL) equation, which we consider over 1D, 2D, and 3D cuboids with homogeneous Neumann and Dirichlet BC. Next we consider a reaction diffusion system from [GKS00] on a circular domain and with Robin BC. As a consequence, Hopf bifurcation points corresponding to modes with an angular dependence are double, which leads to the bifurcation of clockwise and anti-clockwise rotating (spiral) waves, and of standing waves.<sup>1</sup> The third example is a Brusselator system from [YDZE02], which shows interesting interactions between Turing branches and Turing–Hopf branches. As a non-dissipative example we treat the canonical system for an optimal control problem (see also [Uec17b]) of “optimal pollution”. This is still of the form (1.1), but is ill-posed as an initial value problem, since it includes “backward diffusion”. Nevertheless, we continue steady states and obtain Hopf bifurcations and branches of periodic orbits.

Additionally, we give two tutorial examples for Hopf bifurcations in systems with continuous symmetries, namely a reaction-diffusion problem with mass conservation, and a Kuramoto–Sivashinsky equation with translational and boost invariance. Such symmetries were not considered systematically in [Uec17a]. They require phase conditions, first for the reliable continuation of steady states and detection of (Hopf or steady) bifurcations, which typically lead to the coupling of (1.1) with algebraic equations. To compute Hopf branches, i.e., branches of time periodic orbits we then also need to set up suitable phase conditions for the Hopf orbits. See also [RU17a], where two more examples of Hopf orbits for systems with symmetries are considered, namely modulated (traveling) fronts in a combustion model, and breathing pulses in a FitzHugh–Nagumo type system.

The user interfaces for Hopf bifurcations reuse the standard `pde2path` setup and no new user functions are necessary, except for the case of symmetries, which requires one additional user function. For the basic ideas of continuation and bifurcation in steady problems we refer to

---

<sup>1</sup>Our default branch switching so far only deals with simple Hopf bifurcation points; for the double Hopf points we use an ad hoc modification.

[UWR14] (and the references therein), for `pde2path` installation hints and review of data structures to [dWDR<sup>+</sup>17], for the `OOPDE` setting in `pde2path` and a general soft introduction to [RU17b], and for the algorithms implemented in the `hopf` library of `pde2path` to [Uec17a]. Thus, here we concentrate on how to use these routines.

The basic setup of all demos is similar. Each demo directory contains:

- Function files named `*init.m` for setting up the geometry and the basic `pde2path` data, where `*` stands for the problem, e.g., `cGL` (and later `rot`, `brussel`, ...).
- Main script files, such as `cmds*d.m` where `*` stands for the space dimension.
- Function files `sG.m` and `sGjac.m` for setting up the rhs of the equation and its Jacobian. Most examples are 2nd order semilinear, i.e., of the form  $\partial_t u = -G(u) = D\Delta u + f(u)$  with diffusion matrix  $D \in \mathbb{R}^{N \times N}$ , and we put the 'nonlinearity'  $f$  (i.e., everything except diffusion) into a function `nodalf.m`, which is then called in `sG.m`, but also in mesh-adaption and in normal form computations. An exception is the KS equation, see §5.2.
- a function `oosetfemops.m` for setting up the system matrices, except for the demo `rot` where we use the old `pdetoolbox` setup.
- A few auxiliary functions, for instance small modifications of the basic plotting routine `hopplot.m` from the `hopf` library, which we found convenient to have problem adjusted plots.
- Some auxiliary scripts `auxcmds.m`, which contain commands, for instance for creating movies or for mesh-refinement, which are not genuinely related to the Hopf computations, but which we find convenient for illustrating either some mathematical aspects of the models, or some further `pde2path` capabilities, and which we hope the user will find useful as well.
- For the demo `pollution` (an optimal control problem) we additionally have the functions `polljcf.m`, which implements the objective function, and `pollbra.m`, which combines output from the standard Hopf output `hobra.m` and the standard OC output `ocbra.m`.

In all examples, the meshes are chosen rather coarse, to quickly get familiar with the software. We did check for all examples that these coarse meshes give reliable results by running the same simulations on finer meshes, without qualitative changes. We give hints about the timing and indications of convergence, but we refrain from a genuine convergence analysis. In some simulations (demo `rot`, and `cGL` in 3D and `brussel` in 2D) we additionally switch off the on the fly computation of Floquet multipliers and instead compute the multipliers a posteriori at selected points on branches. Computing the multipliers simultaneously is possible as well, but may be slow. Nevertheless, even with the coarse meshes some commands, e.g., the continuation of Hopf branches in 3+1D (with about 120000 total degrees of freedom), may take several minutes. All computational times given in the following are from an i7 laptop with Linux Mint 18 and Matlab 2016b.

In §2 to §4 we explain the example implementations for the four examples from [Uec17a], and thus in passing also the main data-structures and functions from the `hopf` library, and in §5 we explain the setup for the two systems with symmetries and hence constraints. For the unconstrained theory and background on the first four example problems we refer to [Uec17a], but for easier reference and to explain the setup with constraints we also give the pertinent formulas in Appendix A. Appendix B then contains an overview of the functions in the `hopf` library and of the relevant data structures for quick reference. For comments, questions, and bugs, please mail to [hannes.uecker@uni-oldenburg.de](mailto:hannes.uecker@uni-oldenburg.de).

**Acknowledgment.** Many thanks to Francesca Mazzia for providing TOM [MT04], which was essential help for setting up the `hopf` library; to Uwe Prüfert for providing `OOPDE`; to Tomas Dohnal, Jens Rademacher and Daniel Wetzel for some testing of the Hopf examples; to Daniel Kressner for `pqzschur`; and to Dieter Grass for the cooperation on distributed optimal control problems, which was one of my main motivations to implement the `hopf` library.

## 2 The cGL equation as an introductory example: Demo cGL

We consider the cubic-quintic complex Ginzburg–Landau equation

$$\partial_t u = \Delta u + (r + i\nu)u - (c_3 + i\mu)|u|^2 u - c_5|u|^4 u, \quad u = u(t, x) \in \mathbb{C}, \quad (2.1)$$

with real parameters  $r, \nu, c_3, \mu, c_5$ . In real variables  $u_1, u_2$  with  $u = u_1 + iu_2$ , (2.1) can be written as the 2-component system

$$\partial_t \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} \Delta + r & -\nu \\ \nu & \Delta + r \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} - (u_1^2 + u_2^2) \begin{pmatrix} c_3 u_1 - \mu u_2 \\ \mu u_1 + c_3 u_2 \end{pmatrix} - c_5 (u_1^2 + u_2^2)^2 \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}. \quad (2.2)$$

We set  $c_3 = -1, c_5 = 1, \nu = 1, \mu = 0.1$ , and use  $r$  as the main bifurcation parameter. Considering (2.2) on, e.g., a (generalized) rectangle

$$\Omega = (-l_1\pi, l_1\pi) \times \dots \times (-l_d\pi, l_d\pi) \quad (2.3)$$

with homogeneous Dirichlet BC or Neumann BC, or with periodic BC, we can explicitly calculate all Hopf bifurcation points from the trivial branch  $u = 0$ , located at  $r = |k|^2 := k_1^2 + \dots + k_d^2$ , with wave vector  $k \in \mathbb{Z}/(2l_1) \times \dots \times \mathbb{Z}/(2l_d)$ . Moreover, for periodic BC we can also explicitly compute the bifurcating (traveling wave) time periodic branches and their Floquet multipliers, see [Uec17a], but for simplicity here we use Neumann and Dirichlet BC, and thus focus on standing waves. For these we still have the formula  $r = |k|^2$  for the HBPs, but we lose the explicit solution formulas, except for the spatially homogeneous branch for  $k = 0$  with Neumann BC. In summary, (2.2) is a nice toy problem to validate and benchmark our routines.

The cGL demo directory consists, as noted above, of some function files to set up and describe (2.2), some script files to run the computations, and a few auxiliary functions and scripts to explain additional features, or, e.g., to produce customized plots. An overview is given in Table 1.

Table 1: Scripts and functions in `hopfdemos/cGL`. Treating the 1D, 2D and 3D cases in one directory, the only dimension dependent files are the scripts, and the function `cGLinit`. The lower block contains auxiliary functions and scripts which are not needed for the Hopf computations, but which illustrate additional `pde2path` features.

script/function	purpose,remarks
<code>cmds*d</code>	main scripts; for $* = 1, 2, 3$ , respectively, which are all quite similar, i.e., mainly differ in settings for output file names. Thus, only <code>cmds1d</code> is discussed in some detail below.
<code>p=cGLinit(p,lx,nx,par,ndim)</code>	init function, setting up parameters and function handles, and, as the only space dimension $d=\text{ndim}$ dependent points, the domain.
<code>p=oosetfemops(p)</code>	set FEM matrices (stiffness K and mass M)
<code>r=sG(p,u)</code>	encodes $G$ from 2.2 (including the BC)
<code>f=nodalf(p,u)</code>	the 'nonlinearity' in (2.2), i.e., everything except $D\Delta u$ .
<code>Gu=sGjac(p,u)</code>	Jacobian $\partial_u G(u)$ of $G$ .
<code>auxcmds1</code>	script, auxiliary commands, illustrating stability checks by time-integration, and a posteriori computation of Floquet multipliers
<code>auxcmds2</code>	script, auxiliary commands, illustrating (adaptive) mesh-refinement by either switching to natural parametrization, or via <code>hopftref</code>
<code>cmds1dconv</code>	script showing some convergence of periods in 1D for finer $t$ discretization
<code>plotana1(k2,rstep,ps,sw,ms)</code>	plot analytical Hopf-branch for cGL for comparison with numerics, used in <code>cmds1d.m</code> , calls <code>mvu=anafloq(rvek,s)</code>
<code>homov2d, homov3d</code>	auxiliary functions to generate movies of Hopf orbits

As main functions we have

- `cGLinit.m`, which (depending on the spatial dimension) sets up the domain, mesh, boundary conditions, and sets the relevant function handles `p.fuha.sG=@sG` and `p.fuha.sGjac=@sGjac` to encode the rhs of (2.2);
- `sG.m` and `sGjac`, which encode (2.2) and the associated Jacobian of  $G$ ;

Then we have three script files, `cmds*d.m`, where  $*$ =1,2,3 stands for the spatial dimension. These are all very similar, i.e., only differ in file names for output and some plotting commands, but the basic procedure is always the same:

1. call `cGLinit`, then `cont` to find the HBPs from the trivial branch  $u \equiv 0$ ;
2. compute branches of periodic orbits (including Floquet multipliers) by calling `hoswibra` and `cont` again, then plotting.

Listings 5-4 discuss the dimension independent (function) files. We use the OOPDE setup, and thus we refer to [RU17b] for a general introduction concerning the meaning of the stiffness matrix  $K$ , the mass matrix  $M$  and the BC matrices  $Q$  (and  $G_{BC}$ , not used here), and the initialization methods `stanpdeo*D`,  $*$  = 1,2,3, setting up an OOPDE object which contains the geometry and FEM space, and the methods to assemble the system matrices.

```
function p=oosetfemops(p) % FEM operators for cGL
grid=p.pdeo.grid; % just a shorthand
[K,M,~]=p.pdeo.fem.assema(grid,1,1,1); % assemble 'scalar' K and M
[Q,~,~]=p.pdeo.fem.assemb(grid); % BC matrix
5 sf=p.nc.sf; % stiff spring factor to implement DBC
N=sparse(grid.nPoints, grid.nPoints); % 0-matrix, another shorthand
p.mat.K=[[K+sf*Q N];[N K+sf*Q]]; % build 2-comp. system K
p.mat.M=kron([1,0];[0,1],M); % and M
```

Listing 1: `cGL/oosetfemops.m`. This sets the stiffness matrix  $K$ , the mass matrix  $M$ , and the BC matrix  $Q$  for (2.2); see [RU17b, §1] for the meaning of these matrices.  $K \in \mathbb{R}^{n_p \times n_p}$  in line 2 is the 'scalar' (i.e., one component) Neumann Laplacian, while  $M$  is the scalar mass matrix. Similarly,  $Q \in \mathbb{R}^{n_p \times n_p}$  in line 3 is a BC matrix for one component, and its content depends on the BC set in lines 7, 9 and 11 of `cGLinit`. In line 4 of `oosetfemops` we create a zero matrix for convenience, and in line 5 we then set up the FEM matrices for the *system* (2.2). Here both diagonal blocks of `p.mat.K` are equal, because so are the diffusion constants for both components in (2.2) and the BC we consider. However, this setup is quite flexible to implement also more complicated differential operators, including off-diagonal blocks ('cross diffusion'), first order differential operators, and different BC in different components.

```
function r=sG(p,u) % compute pde-part of residual
f=nodalf(p,u); r=p.mat.K*u(1:p.nu)-p.mat.M*f;
```

Listing 2: `cGL/sG.m`. Given  $K$  and  $M$  from `oosetfemops`, we only need to compute the 'nonlinearity'  $f$ , which we outsource to `nodalf`, see Listing 3, and then compute the rhs  $G$  (or residual  $r$ ) as  $G(u) = Ku - Mf$ . Note that the BC are already included in `p.mat.K` via line 7 of `oosetfemops`.

```
function f=nodalf(p,u) % nonlinearity for cGL
u1=u(1:p.np); u2=u(p.np+1:2*p.np); par=u(p.nu+1:end); % extract fields
r=par(1); nu=par(2); mu=par(3); c3=par(4); c5=par(5); % and parameters
ua=u1.^2+u2.^2; % aux variable |u|^2
5 f1=r*u1-nu*u2-ua.*(c3*u1-mu*u2)-c5*ua.^2.*u1;
f2=r*u2+nu*u1-ua.*(c3*u2+mu*u1)-c5*ua.^2.*u2;
f=[f1;f2];
```

Listing 3: `cGL/nodalf.m`. The 'nonlinearity' (which includes linear terms, i.e., everything except the diffusion terms)  $f$  from (2.2). We extract the two components  $u_1$  and  $u_2$ , and the parameters from  $u$ , introduce an auxiliary variable  $ua = |u|^2$ , and write down the components of  $f$  in a standard *Matlab* way.

```
function Gu=sGjac(p,u) % Jacobian for cGL
n=p.np; [f1u,f1v,f2u,f2v]=njac(p,u); % the main work
Fu=[[spdiags(f1u,0,n,n),spdiags(f1v,0,n,n)]; % put partial derivatives
[spdiags(f2u,0,n,n),spdiags(f2v,0,n,n)]]; % into (sparse) Jac
5 Gu=p.mat.K-p.mat.M*Fu; % multiply by M and add Laplacian
end
```

```

function [f1u,f1v,f2u,f2v]=njac(p,u) % local (no spat.derivatives) Jacobian
u1=u(1:p.np); u2=u(p.np+1:2*p.np); par=u(p.nu+1:end); % extract fields
10 r=par(1); nu=par(2); mu=par(3); c3=par(4); c5=par(5); % and parameters
ua=u1.^2+u2.^2;
f1u=r-2*u1.*(c3*u1-mu*u2)-c3*ua-4*c5*ua.*u1.^2-c5*ua.^2;
f1v=-nu-2*u2.*(c3*u1-mu*u2)+mu*ua-4*c5*ua.*u1.*u2;
f2u=nu-2*u1.*(c3*u2+mu*u1)-mu*ua-4*c5*ua.*u1.*u2;
15 f2v=r-2*u2.*(c3*u2+mu*u1)-c3*ua-4*c5*ua.*u2.^2-c5*ua.^2;
end

```

Listing 4: cGL/sGjac.m. Similar to sG, the main problem specific part is  $\partial_u f$ , put into njac, the implementation of which follows immediately from nodalf. Gu in line 5 is then rather generic.

```

function p=cGLinit(p,lx,nx,par,ndim) % (generic) init routine for cGL problem
p=stanparam(p); screenlayout(p); % set standard parameters and screenlayout
p.fuha.sG=@sG; p.fuha.sGjac=@sGjac; p.sw.jac=1; % rhs and Jac
p.nc.neq=2; p.nc.ilam=1; p.fuha.outfu=@hobra; % number of eq, cont-param, output
5 switch ndim % domain and BC, depending on spatial dim
case 1; pde=stanpdeo1D(lx,2*lx/nx); p.vol=2*lx; p.x0i=10; % index for ts-plot
bc=pde.grid.neumannBC('0'); % BC
case 2; pde=stanpdeo2D(lx,lx/2,2*lx/nx); p.vol=2*lx^2; p.x0i=30;
bc=pde.grid.robinBC('1','0');
10 case 3; pde=stanpdeo3D(lx,lx/2,lx/4,2*lx/nx); p.vol=0.5*lx^3; p.x0i=200;
bc=pde.grid.robinBC('1','0');
p.plot.ng=20; p.plot.lev=[-0.1 0.1]; % 3D specific plot settings
p.plot.levc={'blue','red'}; p.plot.alpha=0.5;
end
15 pde.grid.makeBoundaryMatrix(bc); p.nc.sf=1e3; p.pdeo=pde; % OOPDE setting of BC
p.sw.sfm=-1; p.np=pde.grid.nPoints; p.nu=p.np*p.nc.neq; p.sol.xi=1/p.nu;
p=setfemops(p); % setfemops calls oosetfemops in problem dir
u=0*ones(p.np,1); v=u; p.u=[u;v; par]; % initial guess (here trivial) and pars
p.usrlam=[-0.25 -0.2 -0.1 0 0.5 1 2 3]; % user-vals for output
20 p.plot.cm=hot; p.plot.bpcmp=9; % colormap for soln plot, comp for branch-plot
[p.u,res]=nloop(p,p.u); fprintf('first res=%g\n',res); % start-point for cont
p.file.smod=10; p.sol.ds=0.1; p.nc.dsmax=0.5; % saving, stepsize, max stepsize
p.sw.bifcheck=2; p.nc.neig=20; % method for bifcheck, and # Evals used
p.nc.mu1=0.5; % be relaxed about possible bif-detection

```

Listing 5: cGL/cGLinit.m, which collects some typical initialization commands.  $p=\text{stanparam}(p)$  in line 2 sets the pde2path controls, switches and numerical constants to standard values; these can always be overwritten afterwards, and some typically are. In line 3 set the function handles to the rhs and its Jacobian, and similarly in line 4 we (re)set the output function handle to `hobra`, which can be used as a standard output when Hopf bifurcations are expected. In lines 5-14, depending on the spatial dimension, we create a 1D, 2D or 3D OOPDE objects, essentially consisting of the domain, the FEM setup and the boundary condition. In 1D, this is the interval  $(-lx, lx)$  with mesh width  $lx/nx$ , and homogeneous Neumann BC. In line 15 we finish this by preparing the associated BC matrices, and afterwards we put this PDE-object, the number of grid points, and the associated norm weight  $\xi$  into  $p$ . Calling `setfemops` in line 17 then immediately refers to `oosetfemops`, see Listing 1. In line 18 we initialize the solution vector (here with the explicitly known trivial solution  $u = 0$  and append the parameters. In the remainder of `acinit` we set some additional controls, mostly explained by the comments. We only remark that  $p.\text{sw.bifcheck}=2$  in line 23 tells `pde2path` to use algorithm HD2 [Uec17a, §2.1] to detect bifurcations, by computing  $n_{\text{eig}}=p.\text{nc.neig}=20$  eigenvalues near 0. This is a suitable choice since  $\partial_u G(0)$  has no real eigenvalues.  $p.\text{nc.mu1}$  in line 24 refers to  $\mu_1$  from [Uec17a, Remark 2.2]. Finally,  $p.\text{vol}$  in lines 6,8 and 10 is used in the norm (2.4), and  $p.\text{x0i}$  is a point index for plotting the time-series  $t \mapsto u(t, x_{p.\text{x0i}})$ .

## 2.1 1D

In 1D we use Neumann BC, and  $n_x = 31$  spatial, and (without mesh-refinement)  $m = 21$  temporal discretization points. Listings 6 and 7 shows the main script file `cmds1d.m` for 1D computations

(with some omissions wrt to plotting), while Fig. 1 shows some output generated by running `cmds1d`. The norm in (a) is

$$\|u\|_* := \|u\|_{L^2(\Omega \times (0,T), \mathbb{R}^N)} / \sqrt{T|\Omega|}, \quad (2.4)$$

which is our default for plotting of Hopf branches. During the continuation the default plotting routine `hoplot` also plots the time-series  $t \mapsto u_1(x_0, t), u_2(x_0, t)$  for some mesh point  $x_0$ , selected by the index `p.hopf.x0i`, which is set in `cGLinit` (see also Fig. 1(b)). The simulations run in less than 10 seconds per branch, but the rather coarse meshes lead to some inaccuracies. For instance, the first three HBPs, which analytically are at  $r = 0, 1/4, 1$ , are obtained at  $r = 6 \cdot 10^{-5}, 0.2503, 1.0033$ , and (b) also shows some visible errors in the period  $T$ . However, these numerical errors quickly decay if we increase  $n_x$  and  $m$ , and runtimes stay small.

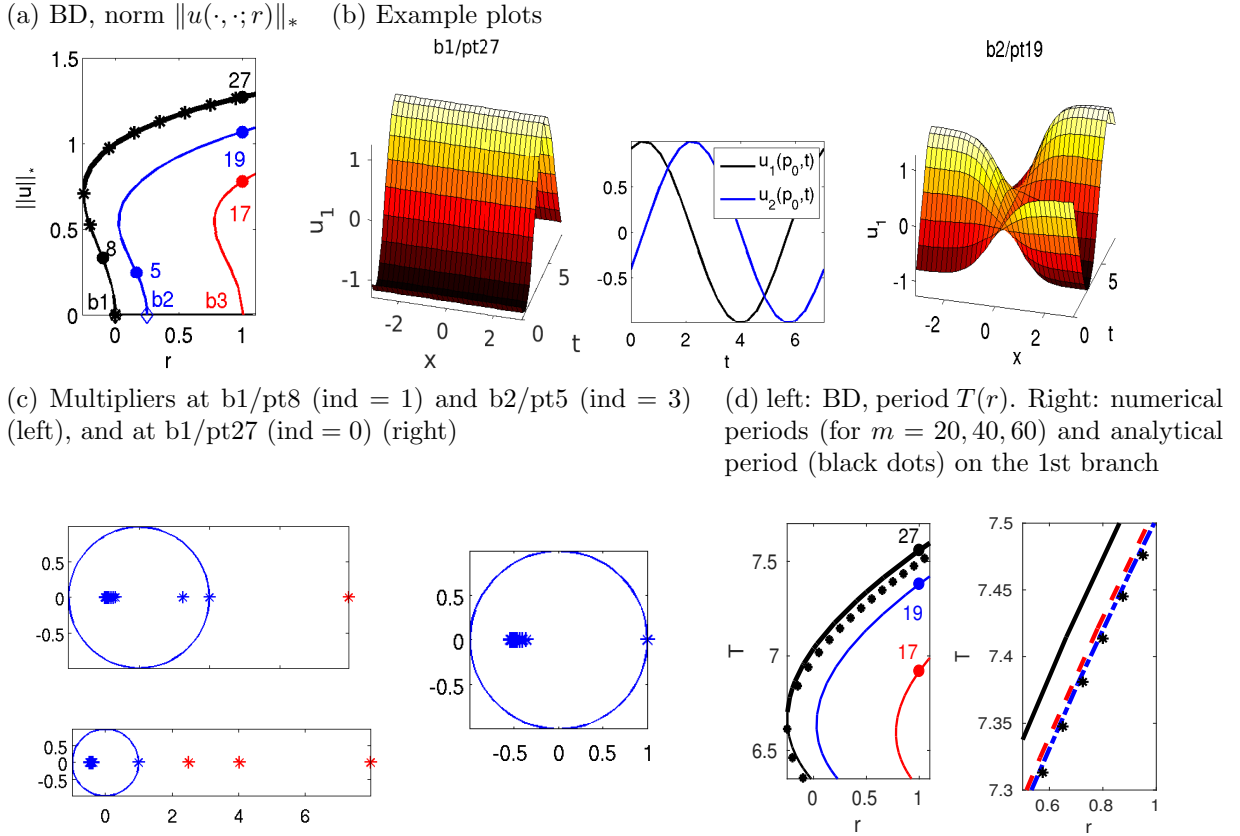


Figure 1: Selected outputs from `cmds1d.m`, i.e., numerical bifurcation diagrams, example plots and (leading 20) Floquet multipliers for (2.2) on the domain  $\Omega = (-\pi, \pi)$  with Neumann BC, 30 grid-points in  $x$ . Parameters  $(\nu, \mu, c_3, c_5) = (1, 0.1, -1, 1)$ , hence bifurcations at (restricting to the first three branches)  $r = 0$  ( $k = 0$ , spatially homogeneous branch, black),  $r = 1/4$  ( $k = 1/2$ , blue) and  $r = 1$  ( $k = 1$ , red). The thick part of the black line in (a) indicates the only stable periodic solutions. The black dots in (a) and (d) are from the (spatially homogeneous) analytical solution, see [Uec17a]. For  $m = 20$  there is a visible error in  $T$ . The right panel of (d) shows the numerical  $T$  for different  $m$  ( $m = 20$  black,  $m = 40$  red-dashed,  $m = 60$  blue-dotted), which illustrates the convergence of the numerical solution towards the analytical solution (??). Similarly, the periods also converge on the other branches (see `cmds1dconv.m`). The second plot in (b) shows a time series at the point `p.hopf.x0i` from b1/pt27. See also Fig. 2(b) for a plot of b3/pt17.

As usual we recommend to run `cmds1d` cell-by-cell to see the effect(s) of the individual cells.

```
close all; format compact; keep pphome;
% cell1: init, and continuation of trivial branch
ndim=1; dir='hom1d'; p=[]; lx=pi; nx=30; % domain size and spat.resolution
par=[-0.05; 1; 0.1; -1; 1]; % r nu mu c3 c5
```

```

5 p=cGLinit(p,lx,nx,par,ndim); p=setfn(p,dir); % initialization and dirname
p=cont(p,20); % continuation of (here) trivial branch, incl. bif-detec
%% cell2: first 2 Hopf branches, run arclength from the start
para=4; ds=0.1; figure(2); clf; aux=[]; %aux.tl=60;
for bnr=1
10     switch bnr
        case 1; p=hoswibra('hom1d','hpt1',ds,para,'1db1',aux); nsteps=30;
        case 2; p=hoswibra('hom1d','hpt2',ds,para,'1db2'); nsteps=20;
    end
    p.hopf.jac=1; p.nc.dsmax=0.5; p.hopf.xi=0.05; p.file.smod=5; p.sw.verb=0;
15    p.hopf.flcheck=1; % switch for Floquet-comp: 0: off, 1:floq, 2: floqps
    bw=1; beltol=1e-6; belimax=5; % border-width, bel-parameters
    droptol=1e-3; AMGmaxit=50; % ilupack parameters (only needed if AMG=1)
    AMG=0; % set AMG=1 if ilupack available
    if ~AMG; p=setbel(p,bw,beltol,belimax,@lss); % use BEL without ilupack
20    else p=setbel(p,0,beltol,belimax,@lssAMG); p=setilup(p,droptol,AMGmaxit);
    end
    t1=tic; p=cont(p,nsteps); toc(t1)
end

```

Listing 6: cGL/cmds1d.m (first two cells). In cell 1 we initialize the problem and continue the trivial branch (with standard settings) to find the HBPs. In cell 2 we then compute the first 2 bifurcating Hopf branches in the arclength setting. See Appendix B for comments on `hoswibra`, which sets all the data structures for periodic orbit continuation and of course an initial guess, and thus is the main routine here. In line 15 we switch on the Floquet computation with `floq`, see §2.2. In line 16 we set parameters for the (optional) bordered elimination linear system solver `lssbel`, see [Uec17a, Remark 2.3], and in line 17 for the preconditioned ilupack solver [Bol16] `lssAMG` as an inner solver for `lssbel`. This is optional, and controlled by the switch `AMG` in line 18. See lines 19,20 for the convenience functions to switch on these solvers and to set parameters. For the present 1D problem, both `lssAMG` or just `lss` are roughly equally fast, but for larger scale problems `lssAMG` is significantly faster. In any case, without `ilupack`, `lssbel` gives a significant speedup over `lss` for bordered systems, see also [UW17a] for a tutorial on these solvers.

```

%% cell3: on branch 3, use tomsol for initial steps, then switch to arclength
ds=0.2; p=hoswibra('hom1d','hpt3',ds,3,'1db3');
p.hopf.xi=0.05; p.hopf.jac=1; p.nc.dsmax=0.25;
p.hopf.tom.AbsTol=1e-4; p.hopf.tom.RelTol=1e-3; % tolerances for TOM
5 p=cont(p,5); % do 5 steps in natural parametrization
p.sw.para=4; % then switch to arclength
if ~AMG; p=setbel(p,bw,beltol,belimax,@lss); % use BEL without ilupack
else p=setbel(p,bw-1,beltol,belimax,@lssAMG); p=setilup(p,droptol,AMGmaxit); end
tic; p=cont(p,15); toc
10 %% cell4: plot BD, amplitude, L^2
bpcmp=9; wnr=3; figure(wnr); clf
plotbra('hom1d',3,bpcmp,'lsw',4); % label only HBPs
plotbra('1db1', 3,bpcmp,'lab',[8, 27]); % ... some omissions
%% cell 5: plot BD, T
15 bpcmp=6; wnr=3; figure(wnr); clf;
plotbra('1db1',3,bpcmp, 'lab',27, 'fp',1); % ...
%% cell 6: plot solns
hoplotf('1db1','pt27',1,1); figure(1); title('1db1/pt27');

```

Listing 7: cGL/cmds1d.m (continued). In cell 3 we do the initial steps for the third Hopf branch in natural parametrization, which gives a refinement of the  $t$ -mesh by TOM from  $m = 21$  to  $m = 41$  (here uniform due to the harmonic nature of the time-dependence). In line 6 we then switch to arclength and proceed as before. The remaining cells illustrate (with some omissions) the plot of bifurcation diagrams and solutions. Since in `cGLinit` we set `p.fuha.outfu=@hobra`, i.e., to the standard Hopf branch output, and since we have 5 parameters in the problem, the period  $T$  is at (user-)component 6 of the branch, then follow min and max, and component 9 contains the  $L^2$  norm; see also [dW17] for details on the organization of the branch data and on `plotbra`. Cell 6 (abbreviated) gives solution plots.

Switching to continuation in another parameter works just as for stationary problems by calling `p=hoswiparf(...)`. See Cells 1 and 2 of `cGL/auxcmds1.m` for an example, and Fig. 2(a) for



illustration. Cells 3 and 4 of `auxcmds1.m` then contain examples for mesh-refinement in  $t$ , for which there are essentially two options. The first is to use `p.sw.param=3` and the mesh-adaption of TOM, the second is `hopftref`, see Listing 8.

```

%% cell 1: change continuation param
p=hoswiparf('1db1','pt28','c5b',5,0.1); clf(2); p.usrlam=0.25; p=cont(p,20);
%% cell 2: plot BD and solns
bpcmp=6; pstyle=3; wnr=3; figure(wnr); clf
5 plotbraf('c5b','pt18',3,bpcmp);
  xlabel('c_5'); ylabel('T'); p=loadp('c5b','pt18');
  plotana2(p,0,5,0.25,1,0.05,'b*',2,1); axis tight;
  hoplotf('1db1','pt28',1,1); figure(1); title('c_5=1'); % .. (some omissions)
%% cell3: mesh-refinement in t using TOM:
10 p=loadp('1db2','pt10','1db1ref'); hoplot(p,4,1,1);
  % switch to nat.-parametr., and reset tolerances, then cont
  p=arc2tom(p); p.hopf.tau=[]; p.sol.ds=0.01;
  p.hopf.tom.AbsTol=5e-5; p.hopf.tom.RelTol=5e-4; p=cont(p,5);
  p.sw.param=4; p=tom2arc(p); p=cont(p,5); % switch back to arclength and cont
15 %% cell4: mesh-refinement using hopftref
  p=loadp('1db3','pt17','1db1ref');
  % hogradinf(p); % info about max_t |u-dot| (here useless, since harmonic)
  p=hopftref(p,4); p=cont(p,5); % bisect 5 intervals after t=4 and cont
%% cell5: uniform mesh-refinement
20 p=loadp('1db3','pt17','1db1ref'); fac=2.3;
  p=uhopftref(p,fac); p=cont(p,5); % increase # time-slices by fac, then cont

```

Listing 8: `cGL/auxcmds1.m`. Cells 1 and 2 illustrate switching to another continuation parameter, while cells 3 and 4 give simple examples of mesh-adaption in  $t$ . In cell 3 we use the error estimator build into TOM. In cell 4 we use `hopftref`, which is a purely ad hoc refinement, and which requires a time  $t^*$  where to refine from the user. In some cases, the convenience function `hogradinf(p)`, which inter alia returns the time  $t^*$  where  $\|\partial_t u(\cdot, t)\|_\infty$  is maximal, is useful, though not in this problem since the solutions considered here are rather time harmonic. Note that in contrast to cell 3, or to the routine `meshada` for spatial mesh refinement, neither `hogradinf(p)` nor `hopftref` deal with error estimates in any sense.

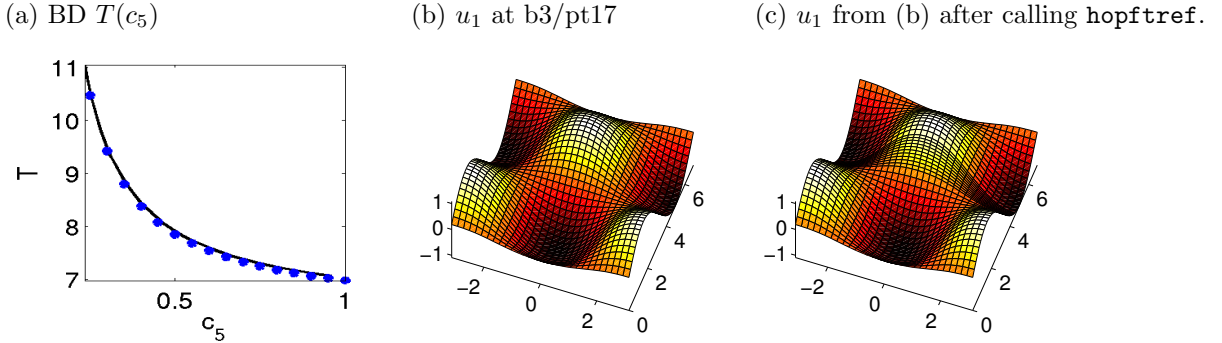


Figure 2: Example outputs from `auxcmds1.m`. (a) Continuing the solution b1/pt28 from Fig. 1(a,b) in  $c_5$ , with comparison to the analytical formula [Uec17a, §3.1]. (b), (c) Solution at b3/pt17 before and after mesh-refinement in  $t$  via `hopftref`, here near  $t^* = 4$ .

## 2.2 Remarks on Floquet multipliers and time integration

For the Floquet multipliers  $\gamma_j$ ,  $j = 1, \dots, n_u$  ( $n_u = Nn_p$  with  $n_p$  the number of spatial discretization points, see (1.3)) we recall from [Uec17a, §2.4] that we have two algorithms for their computation:

- **FA1** (encoded in the function `floq`) computes  $0 \leq \text{p.hopf.nfloq} \leq n_u$  multipliers as eigenvalues of the monodromy matrix  $\mathcal{M}$ .
- **FA2** (encoded in `floqps`) uses a periodic Schur decomposition of the matrices building  $\mathcal{M}$  to compute all  $n_u$  multipliers.

**FA2** is generally much more accurate and robust, but may be slow.<sup>2</sup> See also line 15 in Listing 6. There always is the trivial Floquet multiplier  $\gamma_1 = 1$  associated to translational in  $t$ , and we use  $\text{err}_{\gamma_1} := |\gamma_1 - 1|$  with the numerical  $\gamma_1$  as a measure for the accuracy of the multiplier computation. Furthermore we define the index of a periodic orbit  $u_H$  as

$$\text{ind}(u_H) = \text{number of multipliers } \gamma \text{ with } |\gamma| > 1 \text{ (numerically: } |\gamma| > 1 + \text{p.hopf.ftol}), \quad (2.5)$$

such that  $\text{ind}(u_H) > 0$  indicates instability.

On **b1** in Fig. 1, initially there is one unstable multiplier  $\gamma_2$ , i.e.,  $\text{ind}(u_H) = 1$ , which passes through 1 to enter the unit circle at the fold. On **b2** we start with  $\text{ind}(u_H) = 3$ , and have  $\text{ind}(u_H) = 2$  after the fold. Near  $r = 0.45$  another multiplier moves through 1 into the unit circle, such that afterwards we have  $\text{ind}(u_H) = 1$ , with, for instance  $\gamma_2 \approx 167$  at  $r = 1$ . Thus, we may expect a bifurcation near  $r = 0.45$ , and similarly we can identify a number of possible bifurcation on **b3** and other branches. The trivial multiplier  $\gamma_1$  is  $10^{-12}$  close to 1 in all these computations, using `floq`.

In `cGL/auxcmds2.m` we revisit these multiplier computations, and complement them with time-integration. For the latter, the idea is to start time integration from some point on the periodic orbit, e.g.  $u_0(\cdot) = u_H(\cdot, 0)$ , and to monitor, inter-alia,  $e(t) := \|u(t, \cdot) - u_0(\cdot)\|$ , where by default  $\|\cdot\| = \|\cdot\|_\infty$ . Without approximation error for the computation of  $u_H$  (including the period  $T$ ) and of  $t \mapsto u(\cdot, t)$  we would have  $e(nT) = 0$ . In general, even if  $u_H$  is stable we cannot expect that, in particular due to errors in  $T$  which will accumulate with  $n$ , but nevertheless we usually can detect instability of  $u_H$  if at some  $t$  there is a qualitative change in the time-series of  $e(t)$ .<sup>3</sup> In Fig. 3(a), where we use the smaller amplitude periodic solution at  $r = 0$  for the IC, this happens right from the start. Panel (b) illustrates the stability of the larger amplitude periodic solution at  $r = 0$ , while in (c) the instability of the solution on **h2** at  $r = 1$  manifests around  $t = 30$ , with subsequent convergence to the (stable) spatially homogeneous periodic orbit

```

%% cell 1: plotting of precomputed multipliers
h=plotfloq('1db1','pt8');
%% cell 2: a posteriori compute and plot multipliers
aux=[]; aux.wnr=6; aux.nfloq=10;
5 [gav1,gav2]=floqap('1db2','pt10',aux); axis tight;
%% cell 3: this cell only if percomplex has been mexed
aux.wnr=8; [muv1,muv2]=floqpsap('1db2','pt10',aux);
%% cell 4: time-integration, preparations
p=loadp('1db1','pt10'); hoplot(p,1,1); dir='stab1d1';
10 p.u(1:p.nu)=p.hopf.y(1:p.nu,1); u0=p.u(1:p.nu); p=setfn(p,dir);
ts=[]; t0=0; npp=50; nt=200; pmod=50; smod=5; tsmod=1; nc=0;
%% cell 5: time-integration (repeat if necessary)
[p,t0,ts,nc]=hotintxs(p,u0,t0,ts,npp,nt,nc,tsmod,pmod,smod,@nodalf,1);
figure(4); clf; plot(ts(1,:),ts(2,:)); % plot values at selected point
15 figure(5); clf; plot(ts(1,:),ts(3,:)); % plot difference in norm
%% cell 6: x-t plot; see in ts if there's something interesting after np
% periods, then plot around there
si=3*npp; incr=5; nt=5*npp/smod; wnr=2; cmp=1; vv=[30,70]; nt=35;
tintplot1d(dir,si,incr,nt,wnr,cmp,vv);

```

Listing 9: `cGL/auxcmds2.m`. Cells 1-3 deal with Floquet computations as indicated in the comments. Cells 4-6 deal with time integration. In line 10 we set up  $u(\cdot, t_0)$  from the Hopf-orbit in `1db1/pt8` as an initial condition, and set some parameters. This is used in Cell 5 for time integration via `hotintxs` (see source for documentation), and Cell 6 plots the results, see Fig. 3.

<sup>2</sup> For `floqps` one needs to `mex percomplex.f(F)` in the directory `pqzschur`, see the README file there.

<sup>3</sup> The time integration `hotintxs` takes inter alia the number `npp` of time steps per period  $T$  as argument. Time integration is much faster than the BVP solver used to compute the periodic orbits, and thus `npp` can be chosen significantly larger than the number  $m$  of time-discretization points in the BVP solver. Thus, choosing `npp = 5m` or `npp = 10m` appears a reasonable practice.

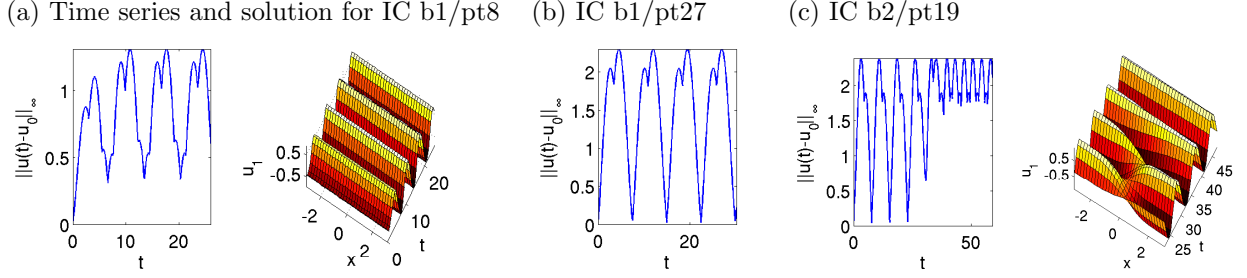


Figure 3: Selected output from `auxcmds2.m`, i.e., stability experiments for (2.2) in 1D. (a) IC h1/pt8, time series of  $\|u(\cdot, t) - u_0\|_\infty$  and  $u_1(x, t)$ , showing the convergence to the larger amplitude solution at the same  $r$ . (b) IC h1/pt27 from Fig. 1, where we plot  $\|u(\cdot, t) - u_0\|_\infty$  for  $t \in [0, 4T]$ , which shows stability of the periodic orbit, and a good agreement for the temporal period under time integration. (c) instability of b2/pt19 from Fig. 1, and again convergence to the solution on the b1 branch. Note that the time-stepping is much finer than the appearance of the solution plots, but we only save the solution (and hence plot) every 100th step, cf. footnote 3.

## 2.3 2D

In 2D we choose homogeneous Dirichlet BC for  $u_1, u_2$ , see lines 8,9 in `cGLinit`, and `oosetfemops.m`. Then the first two HBPs are at  $r_1 = 5/4$  ( $k = (1/2, 1)$ ), and  $r_2 = 2$  ( $k = (1, 1)$ ). The script file `cmds2d.m` follows the same principles as `cmds1d.m`, and includes some time integration as well, and in the last cell an example for creating a movie of a periodic orbits.

Figure 4 shows some results from `cmds2d.m`, obtained on a coarse mesh of  $41 \times 21$  points, hence  $n_u = 1722$  spatial unknowns, yielding the numerical values  $r_1 = 1.2526$  and  $r_2 = 2.01$ . With  $m = 20$  temporal discretization points, the computation of each Hopf branch then takes about a minute. Again, the numerical HBPs converge to the exact values when decreasing the mesh width, but at the prize of longer computations for the Hopf branches. For the Floquet multipliers we obtain a similar picture as in 1D. The first branch has  $\text{ind}(u_H) = 1$  up to the fold, and  $\text{ind}(u_H) = 0$  afterwards, and on b2  $\text{ind}(u_H)$  decreases from 3 to 2 at the fold and to 1 near  $r = 7.2$ . Panel (c) illustrates the 2D analogue of Fig. 3(c), i.e., the instability of the second Hopf branch and stability of the first.

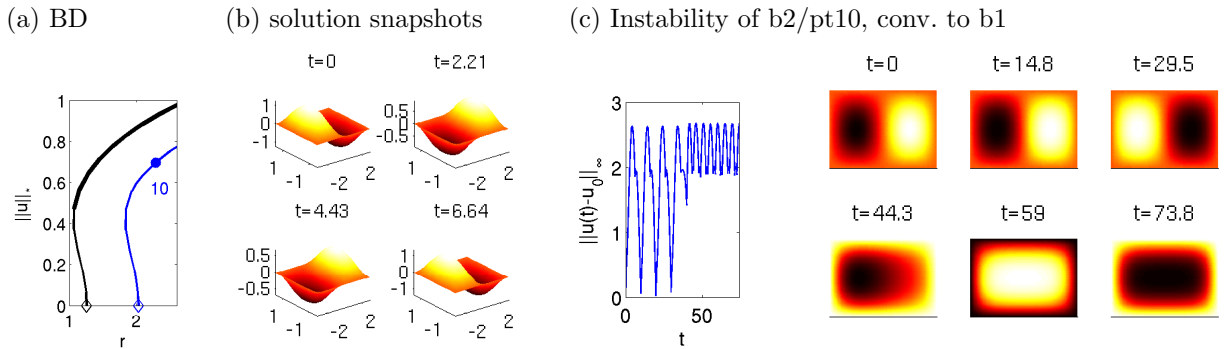


Figure 4: Example plots from `cmds2d.m`. (a) Bifurcation diagrams of the first 2 Hopf branches for (2.2) in 2D. (b) Solution snapshot from b2/pt10, at  $t = 0, \frac{3}{10}T, \frac{6}{10}T, \frac{9}{10}T$ . (c) Time integration starting from (b) ( $t = 0$ ), with convergence to the first Hopf branch.

## 2.4 3D

To illustrate that exactly the same setup also works in 3D, in `cmds3d.m` and Fig. 5 we consider (2.2) over  $\Omega = (-\pi, \pi) \times (-\pi/2, \pi/2) \times (-\pi/4, \pi/4)$ . Here we use a *very* coarse tetrahedral mesh

of  $n_p=2912$  points, thus 5824 DoF in space. Analytically, the first 2 HBPs are  $r_1=21/4$  ( $k = (1/2, 1, 2)$ ) and  $r_2=6$  ( $k = (1, 1, 2)$ ), but with the coarse mesh we numerically obtain  $r_0=5.47$  and  $r_1=6.29$ . Again, this can be greatly improved by, e.g., halving the spatial mesh width, but then the Hopf branches become very expensive. Using  $m = 20$ , the computation of the branches (with 15 continuation steps each) in Fig. 5 takes about 10 minutes, and a call of `floqap` to a posteriori compute the Floquet multipliers about 50 seconds. Again, on b1,  $\text{ind}(u_H)=1$  up to fold and  $\text{ind}(u_H)=0$  afterwards, while on b2  $\text{ind}(u_H)$  decreases from 3 to 2 at the fold and to 1 at the end of the branch, and time integration from an IC from b2 yields convergence to a periodic solution from b1.

The script `cmds2d.m` follows the same principles as the 1D and 2D scripts. In 3D, the “slice plot” in Fig. 5(b), indicated by `p.plot.pstyle=1` should be used as a default setting, while the isolevels in (c) (via `p.plot.pstyle=2`) often require some fine tuning. Additionally we provide a “face plot” option `p.plot.pstyle=3`, which however is useless for Dirichlet BC.

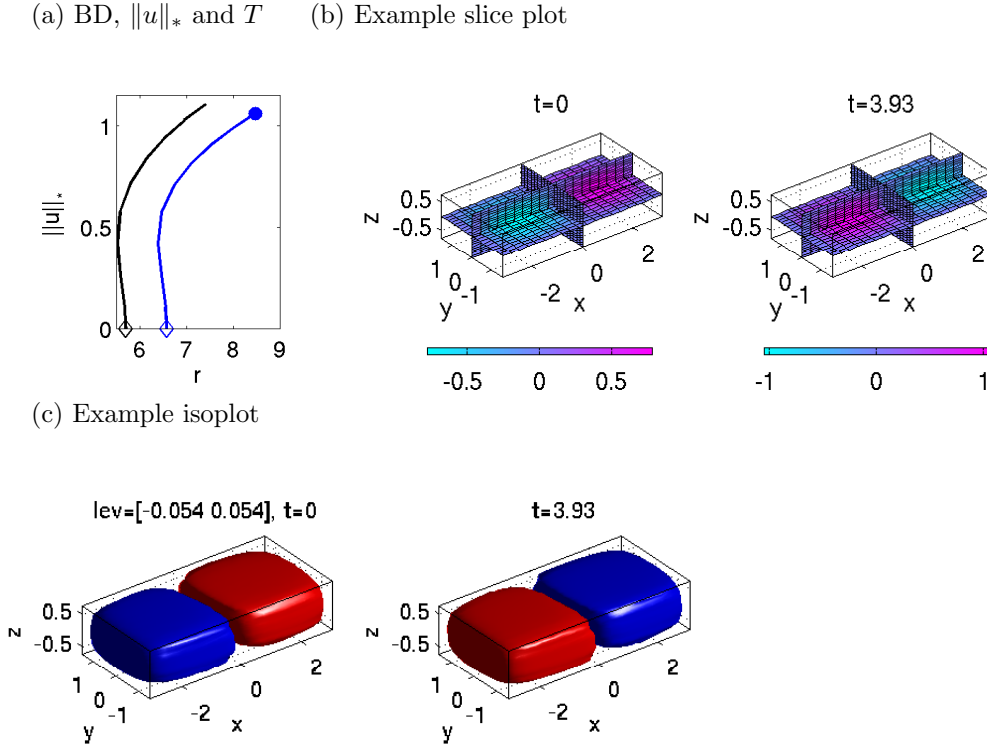


Figure 5: Example plots from `cmds3d.m`. (a) Bifurcation diagram of first 2 Hopf branches for (2.2) in 3D. (b,c) Solution snapshots at  $t = 0$  and  $t = T/2$  for the blue dot in (a); slice-plot in (b), and isolevel plot in (c) with levels  $0.525m_1 + 0.475m_2$  and  $0.475m_1 + 0.525m_2$ , where  $m_1 = \min_{x,t} u_1(x,t)$  and  $m_2 = \max_{x,t} u_1(x,t)$ .

### 3 Two reaction diffusion systems

#### 3.1 Rotating patterns on a disk: Demo rot

In [Uec17a, §3.2] we consider a two-component reaction diffusion system from [GKS00] on the unit disk with somewhat non-standard Robin-BC, namely

$$\begin{aligned} \partial_t u &= d_1 \Delta u + (0.5 + r)u + v - (u^2 + v^2)(u - \alpha v), \\ \partial_t v &= d_2 \Delta v + rv - u - (u^2 + v^2)(v + \alpha u), \end{aligned} \quad (3.1)$$

$$\partial_{\mathbf{n}} u + 10u = 0, \quad \partial_{\mathbf{n}} v + 0.01v = 0, \quad (3.2)$$

where  $\mathbf{n}$  is the outer normal. First (script files `rotcmds_a` and `swcmds.m`) we follow [GKS00] and set  $\alpha = 0$ ,  $d_1 = 0.01$ ,  $d_2 = 0.015$ , and take  $r$  as the main bifurcation parameter. Then (script file `rotcmds_a`) we set  $\alpha = 1$ , let

$$(d_1, d_2) = \delta(0.01, 0.015), \quad (3.3)$$

and also vary  $\delta$  which corresponds to changing the domain size by  $1/\sqrt{\delta}$ . In the end, this gives a model with distinguished bifurcations of spiral waves from the trivial solution  $(u, v) \equiv 0$ .

The eigenfunctions of the linearization around  $(u, v) = (0, 0)$  are build from Fourier Bessel functions

$$\phi_m(\rho, \vartheta, t) = \Re(e^{i(\omega t + m\vartheta)} J_m(q\rho)), \quad m \in \mathbb{Z}, \quad (3.4)$$

where  $(\rho, \vartheta)$  are polar-coordinates, and with, due to the BC (3.2), in general complex  $q \in \mathbb{C} \setminus \mathbb{R}$ . Then the modes are growing in  $\rho$ , which is a key idea of [GKS00] to find modes bifurcating from  $(u, v) = (0, 0)$  which resemble spiral waves near their core. The trivial solution  $(u, v) = (0, 0)$  is stable up to  $r_1 \approx -0.21$  where a Hopf bifurcation with angular wave number  $m = 0$  in (3.4) occurs, and then further Hopf-bifurcations with  $m = 1, 2, 0, 3, \dots$

To discuss the bifurcating branches we need to consider the symmetries of (3.1), (3.2). Additional to time translation, we have the spatial symmetry group  $O(2)$ , acting by rotations and reflections in  $x$ . The modes (3.4) with  $m \neq 0$  have the symmetry of rotating waves (RW), and Hopf bifurcations with  $m \neq 0$  in (3.4) are double. The bifurcating branches are branches of clockwise and counterclockwise RW, and branches of standing waves (SW) given by equal amplitude superpositions of clockwise and counterclockwise RW. This follows from the equivariant Hopf theorem, see, e.g., [GS02, §4.6, 4.7] or [Hoy06, §4.4 and §5.7].

Our default branch switching (A.3) only applies to simple Hopf bifurcations. Nevertheless, if we apply (A.3) at a double Hopf bifurcation in the present example, then it turns out that the initial guess is sufficiently close to a RW for the subsequent Newton loop to converge to this RW. On the other hand, to switch to SW, we use a modified ad hoc ansatz

$$u(t) = u_0 + 2\varepsilon \alpha \operatorname{Re}((z_1 e^{-i\omega_H t} + z_2 e^{i\omega_H t}) \Psi) \quad (3.5)$$

with user provided  $z_1, z_2 \in \mathbb{C}$ , which thus generalizes (A.3) ( $z_1 = 1, z_2 = 0$ ). Using  $z_1 = 1, z_2 = i$  at the HBPs with  $m \geq 1$  yields bifurcation to SW. However, the continuation of SW is more expensive than that of RW, because the phase condition (A.6) fixes the rotational invariance for RW (as time-shifts correspond to rotations), but not for SW. The continuation of SW works, because the FEM discretization destroys the (strict) rotational invariance, but it initially needs small stepsizes due to small eigenvalues, and in the initial continuation steps the orientation of the SW pattern slightly shifts.<sup>4</sup> Steps towards a more systematic treatment of equivariant Hopf bifurcations are intended for the next version of `pde2path`.

Figure 6(a) shows a basic bifurcation diagram for (3.1), (3.2), and (b) and (c) illustrate the difference between rotating and standing waves. Otherwise we refer to [Uec17a, §3.2] for further discussion of these bifurcations, to stability aspects, and to the continuation in other parameters, which lead to more striking rotating spirals. Instead, here restrict to comments on the implementation.

Table 2 lists the pertinent files in `hopfdemos/rot`. The general setup is very similar to `hopfdemos/cGL`, but a difference is that here we use the 'legacy' `pdetoolbox` setting. As a consequence, there is no file `oosetfemops.m`. Instead, in line 5 of `rotinit.m` we define a diffusion tensor, and the system matrices are then generated via the `pde2path` function `setfemops`. Additionally, we set up a function `nbcs.m` defining the boundary conditions. See Listings 10-12. Afterwards, the files `sG.m` and `sGjac.m` encoding (3.1), (3.2), and the script files follow standard rules, where

---

<sup>4</sup>The loss of true rotational invariance can also be seen in the RW, which show small deviations from rigid rotations.

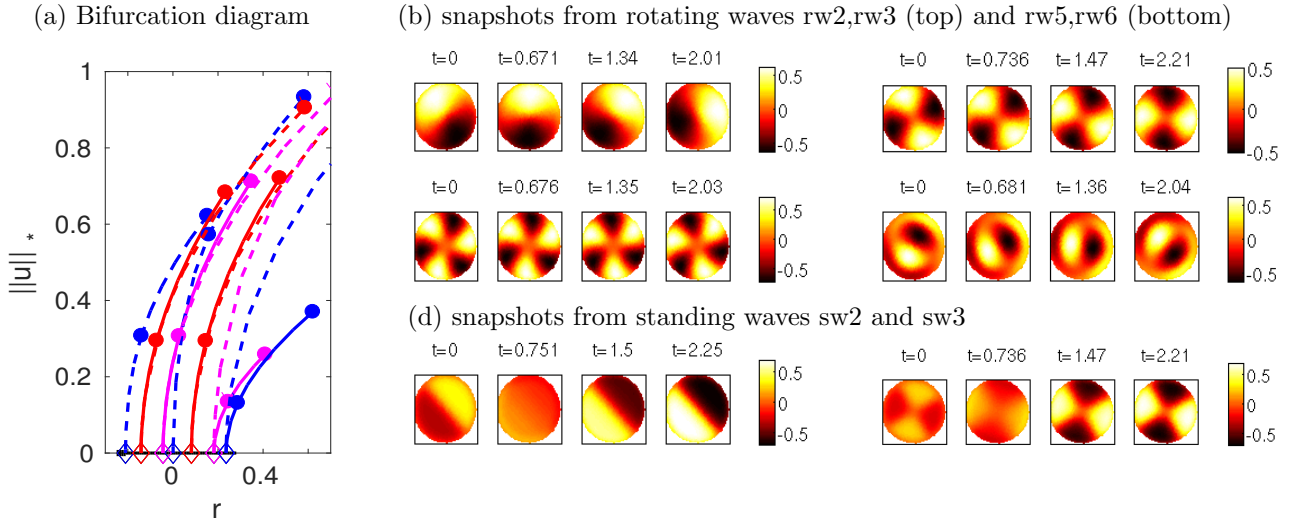


Figure 6: (a) Basic bifurcation diagram for rotating waves (full lines rw2, rw3, rw5, rw6, rw7), and standing waves (dashed lines sw1, ..., sw7) for (3.1), (3.2), 10 continuation steps for each. On sw1 and the RW branches we mark the points 5 and 10. (c) Snapshots of  $u$  from the RW branches at the last points,  $t = 0, T_j/9, 2T_j/9, T_j/3$ , with  $T_j$  the actual period. (c) Snapshots of  $u$  from the RW branches sw2 and sw3. Run `rotcmds_a` and `swcmds.m` to obtain these (and other) plots.

for the plotting we set up some customized functions derived from the default `pde2path` function `hoplotsol`.

Except for different file name settings, the script `swcmds` differs from `rotcmds_a` only in using (3.5) for branch switching. The pertinent command is `aux.z=[1 1i]` (instead of the default setting `aux.z=[]` which means (A.3)), followed by, e.g., `p=hoswibra('tr','hpt2',ds,para,'sw2',aux)`.

Table 2: Scripts and functions in `hopfdemos/rot`.

script/function	purpose,remarks
<code>rotcmds_a</code>	main script, $\alpha = 0$ in (3.1): initialization and finding of HBPs from trivial branch, continuation of first 6 Hopf branches, plotting, and experiments with time-integration
<code>swcmds</code>	script for continuation of branches of standing waves (SW) via a modified branch switching
<code>rotcmds_b</code>	similar to <code>rotcmds_a</code> , but for $\alpha = 1$ in (3.1)
<code>geo=circgeo(r,nx)</code>	defining a circular domain
<code>p=rotinit(p,nx,par)</code>	initialization, as usual
<code>sG, sGjac, nodalf</code>	rhs, Jacobian, and nonlinearity, as usual
<code>nbc</code>	BC for (3.2), using the convenience function <code>gnbc</code>
<code>auxcmds</code>	making movies, using customized plotting from <code>homovplot(...)</code>
<code>hoplotrot, proplot</code>	some additional customized plot commands

```

function p=rotinit(p,nx,par) % init for rot-demo, legacy sfem=1 setting
p=stanparam(p); screenlayout(p);
p.file.dircheck=0; p.nc.ilam=1; p.nc.neq=2; p.fuha.outfu=@hobra;
p.fuha.sG=@sG; p.fuha.sGjac=@sGjac; % rhs
5 p.eqn.c=isoc([0.01,0];[0,0.015]),2,1); % diffusion tensor (for setfemops)
p.eqn.b=0; p.eqn.a=0; % no conv. or linear terms (put these into nodalf)
p.fuha.bc=@nbc; p.fuha.bcjac=@nbc; % BCs
p.mesh.geo=circgeo(1,nx); hmax=2/nx; p=stanmesh(p,hmax);
p.sw.sfem=1; p.vol=2*pi; p.sw.bifcheck=2; p.nc.neig=20;

```



```
10 p=setfemops(p); % here using legacy setfemops, diff
```

Listing 10: `rot/rotinit.m` (first 10 lines). The crucial difference to the other Hopf demos is that this is based on the old `pde toolbox` setting. This leads to changes in lines 5-8, i.e., the setup of the tensors and BC needed by `setfemops` in line 10 (which does *not* call a function `oosetfemops` if `p.sw.sfem`  $\neq -1$ ).

```
function geo=circgeo(r,nx)
td=linspace(0,2*pi,nx); geo=polygong(r*cos(td),r*sin(td));
```

Listing 11: `rot/circgeo.m`. `polygong` is a useful `pde2path` routine (inherited from [Prü16]) for setting up quite arbitrary domains.

```
function bc=NBC(p,u) % BC for model rot
b1=10; b2=0.01; % q-vals in \pa_n u+q*u=0 formulation
c1=p.eqn.c(1); c2=p.eqn.c(end); % diff. constants
b1=b1*c1; b2=b2*c2; % org q-vals need to be multipl. by diff-const.
5 enum=max(p.mesh.e(5,:)); % #ofboundary segments
g=[0;0]; q=[[b1 0]; [0 b2]];
bc=gnbc(p.nc.neq,enum,q,g); % same BC on all bdry segments
```

Listing 12: `rot/nbc.m`, using the 'generalized Neumann BC' convenience function `gnbc`.

### 3.2 An extended Brusselator: Demo brussel

In [Uec17a, §3.2] we consider an example with an interesting interplay between stationary patterns and Hopf bifurcations, and where there are typically many eigenvalues with small real parts, such detecting HBPs with `bifcheck=2` without first using `initeig` for setting a guess for a shift  $\omega_1$  is problematic. The model, following [YDZE02] is an 'extended Brusselator', namely the three component reaction–diffusion system

$$\partial_t u = D_u \Delta u + f(u, v) - cu + dw, \quad \partial_t v = D_v \Delta v + g(u, v), \quad \partial_t w = D_w \Delta w + cu - dw, \quad (3.6)$$

where  $f(u, v) = a - (1+b)u + u^2v$ ,  $g(u, v) = bu - u^2v$ , with kinetic parameters  $a, b, c, d$  and diffusion constants  $D_u, D_v, D_w$ . We consider (3.6) on rectangular domains in 1D and 2D, with homogeneous Neumann BC for all three components. The system has the trivial spatially homogeneous steady state

$$U_s = (u, v, w) := (a, b/a, ac/d),$$

and in suitable parameter regimes it shows co-dimension 2 points between Hopf, Turing–Hopf (aka wave), and (stationary) Turing bifurcations from  $U_s$ . A discussion of these instabilities of  $U_s$  in the  $a - b$  plane is given in [YDZE02] for fixed parameters

$$(c, d, D_u, D_v, D_w) = (1, 1, 0.01, 0.1, 1). \quad (3.7)$$

In our simulations we additionally fix  $a = 0.95$ , and take  $b$  as the primary bifurcation parameter.

For the quite rich bifurcation results, which include primary spatially homogeneous and patterned Hopf bifurcations from  $U \equiv U_s$ , and Turing bifurcations from  $U_s$  followed by secondary Hopf bifurcations, we refer to [Uec17a, §3.2]. Regarding the implementation, Table 3 lists the scripts and functions in `brussel`. Except for the additional component ( $N = 3$  instead of  $N = 2$ ) this is quite similar to `cGL`, with one crucial difference, in particular in 2D, on which we focus in the following discussion.

The linearization of (3.6) around  $U_s$  has many small real eigenvalues. Therefore, the Hopf eigenvalues (with imaginary parts near  $\omega_1 = 1$ ) are impossible to detect by computing just a few eigenvalues close to 0, in particular in 2D, and thus we need a preparatory step `initeig`, which produces a guess for  $\omega_1$  via a Schur complement algorithm, see [Uec17a, §2.1], and the comments in Listing 13.

Table 3: Scripts and functions in `hopfdemos/brussel`.

script/function	purpose,remarks
<code>bru1dcmds</code>	script for 1D, including some time integration; produces [Uec17a, Fig.7(c,d) and Fig.8]
<code>bru2dcmds</code>	script for 2D, including preparatory step <code>initeig</code> for guessing $i\omega$ for Hopf bifurcations, and some time integration
<code>auxcmds1</code>	1D auxiliaries, illustrating spatial mesh refinement on Turing branches
<code>auxcmds2</code>	2D auxiliaries: illustration of problems with many small real eigenvalues
<code>e2rsbru</code>	elements to refine selector, interface to OOPDE's equivalent of <code>pdejumps</code>
<code>evalplot</code>	script for plotting eigenvalues for linearization around spat. homogeneous solution, see [Uec17a, Fig.7(b)].
<code>p=bruinit(p,lx,nx,par,ndim)</code>	initialization as usual, <code>ndim</code> as a parameter to distinguish 1D vs 2D
<code>p=oosetfemops(p)</code>	the FEM operator for (3.6), OOPDE setting
<code>sG, sGjac, nodalf</code>	rhs, Jacobian, and nonlinearity, as usual

```

close all; format compact; keep pphome;
% Cell 1: init hom branch, with INITEIG, then use cont to find bifurcations
Du=0.01; Dv=0.1; Dw=1; c=1; d=1; a=0.95; b=2.75; lx=pi/2;
ndim=2; dir='hom2d'; p=[]; nx=60; par=[a b c d Du Dv Dw];
5 p=bruinit(p,lx,nx,par,ndim); p=setfn(p,dir); p.sw.spcalc=0; p.nc.mu2=0.5e-2;
p=initeig(p,4); p.nc.neig=[3, 3]; % init omv (compute guesses for eval shifts)
p.sw.bifcheck=2; p=cont(p,30); % cont with just 3 evals near 0 and near om1
% Cell 2: Bif from HPs, h1
para=4; ds=0.1; dsmax=0.5; aux=[]; aux.tl=15;
10 p=hoswibra('hom2d','hpt1',ds,para,'2dh1',aux);
p.hopf.ax='unif'; % use same axis for all snapshots from Hopf orbit
p.hopf.fltol=1e-3; % poor accuracy of mu_1 due to coarse mesh in t!
p.hopf.xi=1e-2; p.nc.dsmax=dsmax;
AMG=1; p.sw.verb=3; p.hopf.flcheck=0; % set AMG=1 if ilupack available
15 if ~AMG; p=setbel(p,1,1e-3,5,@lss); % bel significantly faster here
else p=setilup(p,1e-3,50); p.fuha.blss=@lssAMG; end
tic; p=cont(p,10); toc
% Cell 3: steady bif to patterns; cont yields 2ndary Hopf
p=swibra('hom2d','bpt1','2ds1',0.01); p.sw.spcalc=1; p.nc.dsmax=0.01;
20 p.nc.dsmax=0.1; p.nc.ntot=20; p=cont(p,2); % 2 initial steps
p=meshada(p,'ngen',2); % then some mesh-adaption
p.nc.ntot=10; p=pmcont(p); % use pmcont to avoid branch jumping
% Cell 4: 2ndary Hopf bifs from Turing branch
aux=[]; aux.tl=15; ds=0.1; p=hoswibra('2ds1','hpt1',ds,para,'2ds1h1',aux);
25 p.file.smod=1; p.nc.dsmax=0.1;
AMG=1; p.sw.verb=3; p.hopf.flcheck=0;
if ~AMG; p=setbel(p,1,1e-3,5,@lss); % bel significantly faster here
else p=setilup(p,1e-3,50); p.fuha.blss=@lssAMG; end
p.nc.tol=1e-6; tic; p=cont(p,5); toc

```

Listing 13: `brussel/bru2dcmds.m` (first 4 Cells). With `nx=60` in line 4 we obtain a mesh of 961 points. Compared to, e.g., the `cGL` demo, we need the preparatory step in line 6. This produces a (here quite accurate) guess  $0.9375$  for the candidate  $\omega$  for imaginary parts at Hopf bifurcations, which, together with  $\omega_0 = 0$ , is put into `p.nc.eigref`. In Cell 2 we follow the first bifurcating Hopf branch. If `ilupack` is available, then we recommend to set `AMG=1` in line 14. `p.hopf.flcheck=0` switches off multiplier computations, which can also be done a posteriori with `[muv1, muv2, ind]=floqap(pd,varargin)` or, using the periodic Schur decomposition, with `[muv1, muv2, ind]=floqpsap(pd,varargin)`. In cell 3 we follow a bifurcation to a steady spots. Here we first do only 2 steps, and then some mesh-adaption, see Listing 14 for the used error-estimator. This mesh-refinement also makes the further continuation less prone to branch jumping, which is always a problem for pattern forming systems with many branches of pattern close to each other, but for the further steps we nevertheless use `pmcont` [UWR14, §4.2]. This inter alia yields a secondary Hopf bifurcation, which we follow in Cell 4. The remainder of `bru2dcmds` deals with plotting, time-integration, Floquet multipliers and movies.

As indicated in the caption of Listing 13, at the start of the (1D and 2D) Turing branches `s1`



and `2ds1` (see [Uec17a]) we do some adaptive mesh-refinement. See Listing 13 for the 2D case, and Listing 14 for the error estimator for the adaptive mesh refinement. The further BPs and HBPs then obtained are very close to the BPs and HBPs on the coarser mesh, but the resolution of the bifurcating Hopf branches becomes considerably better, with a moderate increase of computation time, which in any case is faster than starting with a uniform spatial mesh yielding a comparable accuracy.

```
function [p,idx]=e2rsbru(p,u) % elements2refine selector as in pdejms
E=zeros(1,p.pdeo.grid.nElements);
par=u(p.nu+1:end); f=nodalf(p,u); a=0;
for i=1:1 % loop over the three components
5 ci=par(4+i); fi=f((i-1)*p.np+1:i*p.np); ui=u((i-1)*p.np+1:i*p.np);
E=E+p.pdeo.errorInd(ui,ci,a,fi); % sum up componentwise error-est.
end
p.sol.err=max(max(E));
idx=p.pdeo.selectElements2Refine(E,p.nc.sig); % select triangles to refine
```

Listing 14: `brussel/e2rsbru.m`. For a general discussion of error estimators in the OOPDE setting we refer to [RU17b]. The only difference is that here we have a 3 component system, and thus we sum up the element wise errors over the components.

## 4 A canonical system from optimal control: Demo pollution

In [Uec16, GU17], `pde2path` has been used to study infinite time horizon distributed optimal control (OC) problems, see also [Uec17b] for a tutorial on OC computations with `pde2path`. As an example for such problems with Hopf bifurcations<sup>5</sup> we consider, following [Wir00], a model in which the states  $v_1 = v_1(t, x)$  and  $v_2 = v_2(t, x)$  are the emissions of some firms and the pollution stock, and the control  $k = k(t, x)$  models the abatement policy of the firms. The objective is to maximize

$$J(v_0(\cdot), k(\cdot, \cdot)) := \int_0^\infty e^{-\rho t} J_{ca}(v(t), k(t)) dt, \quad (4.1a)$$

where  $J_{ca}(v(\cdot, t), k(\cdot, t)) = \frac{1}{|\Omega|} \int_\Omega J_c(v(x, t), k(x, t)) dx$  is the spatially averaged current value function, with local current value  $J_c(v, k) = pv_1 - \beta v_2 - C(k)$ ,  $C(k) = k + \frac{1}{2\gamma} k^2$ , where  $\rho > 0$  is the discount rate. Using Pontryagin's Maximum Principle, the so called canonical system for the states  $v$  and co-states (or Lagrange multipliers or shadow prices)  $\lambda$  can be formally derived as a first order necessary optimality condition, using the intertemporal transversality condition

$$\lim_{t \rightarrow \infty} e^{-\rho t} \int_\Omega \langle v, \lambda \rangle dx = 0. \quad (4.2)$$

The canonical system reads

$$\partial_t v = D\Delta v + f_1(v, k), \quad v|_{t=0} = v_0, \quad (4.3a)$$

$$\partial_t \lambda = -D\Delta \lambda + f_2(v, k), \quad (4.3b)$$

where  $f_1(v, k) = (-k, v_1 - \alpha(v_2))^T$ ,  $f_2(v, k) = (\rho\lambda_1 - p - \lambda_2, (\rho + \alpha'(v_2))\lambda_2 + \beta)^T$ ,  $\partial_n \lambda = 0$  on  $\partial\Omega$ , and where the control  $k$  is given by

$$k = k(\lambda_1) = -(1 + \lambda_1)/\gamma. \quad (4.3c)$$

For convenience we set  $u(t, \cdot) := (v(t, \cdot), \lambda(t, \cdot)) : \Omega \rightarrow \mathbb{R}^4$ , and write (4.3) as

$$\partial_t u = -G(u) := \mathcal{D}\Delta u + f(u), \quad (4.4)$$

<sup>5</sup>which so far could not be found in the systems studied in [Uec16, GU17]

where  $\mathcal{D} = \text{diag}(d_1, d_2, -d_1, -d_2)$ ,  $f(u) = \left( -k, v_1 - \alpha(v_2), \rho\lambda_1 - p - \lambda_2, (\rho + \alpha'(v_2))\lambda_2 + \beta \right)^T$ . Besides the boundary condition  $\partial_{\mathbf{n}}u = 0$  on  $\partial\Omega$  we have the initial condition  $v|_{t=0} = v_0$  (only) for the states. A solution  $u$  of the canonical system (4.4) is called a *canonical path*, and a steady state of (4.4) (which automatically fulfills (4.2)) is called a *canonical steady state (CSS)*. Due to the backward diffusion in  $\lambda$ , and since we only have initial data for half the variables, (4.4) is *not* well posed as an initial value problem. Thus, one method for OC problems of type (4.1) is to first study CSS, and then canonical paths connecting some initial states to some CSS  $u^*$ . This requires the so-called saddle-point property for  $u^*$ , and if this holds, then canonical paths to  $u^*$  can often be obtained from a continuation process in the initial states, see [Uec17b].

A natural next step is to search for time-periodic solutions  $u_H$  of canonical systems, which obviously also fulfill (4.2). The natural generalization of the saddle point property is that

$$d(u_H) := \text{ind}(u_H) - \frac{n_u}{2} = 0, \quad (4.5)$$

i.e., that exactly half of the Floquet multipliers are in the unit circle. In the (low-dimensional) ODE case, there then exist methods to compute connecting orbits to (saddle type) periodic orbits  $u_H$  with  $d(u_H) = 0$ , see [BPS01, GCF<sup>+</sup>08], which require comprehensive information on the Floquet multipliers and the associated eigenspace of  $u_H$ . A future aim is to extend these methods to periodic orbits of PDE OC systems.

However, in [Uec17a, §3.4] we only illustrate that Hopf orbits can appear as candidates for optimal solutions in OC problems of the form (4.1), and that the computation of Floquet multipliers via the periodic Schur decomposition `floqps` can yield reasonable results, even when computation via `floq` completely fails.

For all parameter values, (4.4) has an explicit spatially homogeneous CSS, see [Uec17a], and by a suitable choice of parameters we obtain Hopf bifurcations to spatially homogeneous and spatially patterned time periodic orbits. Concerning the implementation, Table 4 gives an overview of the involved scripts and functions. Since we again use the `OOPDE` setting, and since we restrict to 1D, although (4.4) is a four component system, much of this is very similar to the `cGL` demo in 1D, with the exceptions that: (a) we also need to implement the objective value and other OC related features; (b) similar to `brussel` it is useful to prepare the detection of HBPs via `initeig`; (c) we need to use `flcheck=2` throughout. Thus, in Listings 15-17 we comment on these points, and for plots illustrating the results of running `pollcmds.m` refer to [Uec17a, §3.4].

Table 4: Main scripts and functions in `hopfdemos/pollution`.

script/function	purpose,remarks
<code>pollcmds</code>	main script
<code>p=pollinit(p,lx,nx,par)</code>	init function
<code>p=oosetfemops(p)</code>	set FEM matrices (stiffness K and mass M)
<code>r=pollsG(p,u)</code>	encodes $G$ from (4.4); we avoid implementing the Jacobian here and instead use <code>p.sw.jac=1</code>
<code>f=nodalf(p,u)</code>	nonlinearity, called in <code>sG</code> .
<code>jc=polljcf(p,u)</code>	the (current value) objective function

```
function p=pollinit(p,lx,nx,par) % init-routine for pollution demo
p=stanparam(p); p.nc.neq=4; p.sw.jac=0; % numerical Jac
p.fuha.sG=@pollsG; p.fuha.jcf=@polljcf; % rhs, objective value,
p.fuha.outfu=@pollbra; % customized output (including objective function(s))
```

Listing 15: `pollution/pollinit.m` (first 4 lines). Additional to the rhs, in line 3 we set a function handle to the objective value, as usual for OC problems (see [Uec17b]). Similarly, in line 4 we set `p.fuha.outfu` to a customized branch output, which combines features from the standard Hopf output `hobra` and the standard OC output `ocbra`. We do not set `p.fuha.sGjac` since for convenience here we use numerical Jacobians (`p.sw.jac=0` in line 1). The remainder of `pollinit.m` is as usual.

```

function jc=polljcf(p,u) % current value for pollution
par=u(p.nu+1:end); pr=par(2); vp=par(3); ga=par(5);
y=u(1:p.np); z=u(p.np+1:2*p.np); l1=u(2*p.np+1:3*p.np); % extract soln-components
k=-(1+l1)/ga; c=k+ga*k.^2/2; jc=pr*y-vp*z-c; % compute k, then J

```

Listing 16: pollution/polljcf.m, function to compute the current objective value. Called in pollbra to put the value on the branch (for plotting and other post-processing).

```

%% script for Hopf bif in pollution model Wirl2000, here with diffusion
close all; clear classes; keep pphome
%% cell 1: init and continue trivial branch
p=[]; lx=pi/2; nx=40; par=[0.5 1 0.2 0 300]; % [del, pr, beta, a, ga];
5 p=pollinit(p,lx,nx,par); p=setfn(p,'FSS'); screenlayout(p); p.file.smod=2;
p=initwn(p,2,1); p=initeig(p); p.nc.neig=[5 5]; % find guess for omega_1
p.sw.bifcheck=2; p.sw.verb=2; p.nc.mu2=1e-3; % accuracy of Hopf detection
p.nc.ilam=1; p.sol.ds=0.01; p.nc.dsmax=0.01; p=cont(p,20); % cont of FSS
%% cell2: cont of Hopf branches
10 para=4; ds=0.5; dsmax=1; xi=1e-2; figure(2); clf; aux=[]; aux.tl=25;
for j=1:2
    switch j
        case 1; p=hoswibra('FSS','hpt1',ds,para,'h1',aux); nsteps=15;
        case 2; p=hoswibra('FSS','hpt2',ds,para,'h2',aux); nsteps=25;
15 end
p.hopf.xi=xi; p.hopf.jac=1; p.nc.dsmax=dsmax;
p.file.smod=1; p.hopf.flcheck=2; % use floqps for multipliers
p.usrlam=[0.5 0.6 0.7]; tic; p=cont(p,nsteps); toc
end

```

Listing 17: pollution/pollcmds.m (first 19 lines). In cell 1 we use initeig to generate a guess for  $i\omega_1$  (the Hopf wave number), and set neig to compute 5 eigenvalues near 0 and near  $\omega_1$ . For the computation of multipliers here we need to use floqps, see line 17. The remainder of pollcmds deals with plotting.

## 5 Hopf bifurcation with symmetries

If the PDE (1.1) has (continuous) symmetries, then already for the reliable continuation of steady states it is often necessary to augment (1.1) by  $n_Q$  suitable phase conditions, in the form

$$Q(u, \lambda, w) = 0 \in \mathbb{R}^{n_Q} \quad (5.1)$$

where  $w \in \mathbb{R}^{n_Q}$  stands for the required  $n_Q$  additional active parameters, see [RU17a] for a review. For instance, if (1.1) is spatially homogeneous and we consider periodic BC, then we have a translational invariance, and (in 1D) typically augment (1.1) by the phase condition

$$\langle \partial_x u^*, u - u^* \rangle = 0 \in \mathbb{R}, \quad (5.2)$$

where (for scalar  $u, v$ )  $\langle u, v \rangle = \int_{\Omega} uv \, dx$ , and where  $u^*$  is either a fixed reference profile or the solution from the previous continuation step. We thus have  $n_Q = 1$  additional equations, and consequently must free 1 additional parameter.

Similarly, we must add phase conditions to the computation of Hopf orbits (additional to the phase condition (A.6) fixing the translational invariance in  $t$ ). This is in general not straightforward, since (5.1), with (5.2) as an example, is not of the form  $\partial_t u = Q(u, \lambda)$  and thus cannot simply be appended to (1.1). Instead, the steady phase conditions (5.1) must be suitably modified and explicitly appended to the Hopf system, see (A.9). Examples for the case (5.2) have been discussed in [RU17a, §4], namely the cases of modulated fronts, and of breathers.

Here we give two more examples. The first deals with Hopf orbits in a reaction diffusion system with mass conservation, and the second with Hopf orbits in the Kuramoto-Sivashinsky (KS) equation, where we need two phase conditions, one for mass conservation and one to fix the

translational invariance. For both problems we restrict to 1D; like, e.g., the cGL equation, they both can immediately be transferred to 2D (where for the KS equation we need a third phase condition  $q_3(u) = \langle \partial_y u^*, u \rangle = 0$ , cf. (5.9c)), but the solution spaces and bifurcations then quickly become “too rich”, such that – as often – 2D setups only make sense if there are specific questions to be asked.

### 5.1 Mass conservation: Demo mass-cons

As a toy problem for mass conservation in a reaction diffusion system we consider

$$\partial_t u_1 = \Delta u_1 + d_2 \Delta u_2 + f(u_1, u_2), \quad \partial_t u_2 = \Delta u_2 - f(u_1, u_2), \quad \text{in } \Omega, \quad (5.3)$$

$f(u_1, u_2) = \alpha u_1 - u_1^3 + \beta u_1 u_2$ , with parameters  $d_2, \alpha, \beta \in \mathbb{R}$ , and homogeneous Neumann BC. Then  $m := \frac{1}{|\Omega|} \int u + v \, dx$  is conserved since  $\frac{d}{dt} \int_{\Omega} (u + v) \, dx = \int_{\partial\Omega} \partial_n u_1 + (1 + d_2) \partial_n u_2 \, dS = 0$ . Given a steady state  $(u, v)$  for some fixed  $\alpha, \beta$ , this always comes in a continuous family parametrized by the “hidden” parameter  $m$ . Thus, to study steady states and their bifurcations we use the mass constraint

$$Q(u, \lambda) := \frac{1}{|\Omega|} \int u + v \, dx - m = 0, \quad (5.4)$$

where as usual  $\lambda$  stands for the vector of all parameters. Given this additional equation, we have the differential-algebraic system

$$M\dot{u} = -G(u, \lambda), \quad Q(u, \lambda) = 0, \quad (5.5)$$

and to compute solution branches we need 2 parameters, which we choose as  $\alpha, \beta$ . If we restrict to  $m = 0$ , then we have two explicit branches of homogeneous solutions, namely  $u_2 = -u_1$  and  $u_1 = -\frac{\beta}{2} \pm \sqrt{\frac{\beta^2}{4} + \alpha}$ . We choose the initial point  $(\alpha, \beta) = (1, 1)$ ,  $u_1 = -1/2 - \sqrt{5/4}$ ,  $u_2 = -u_1$  and continue in  $\alpha$ .

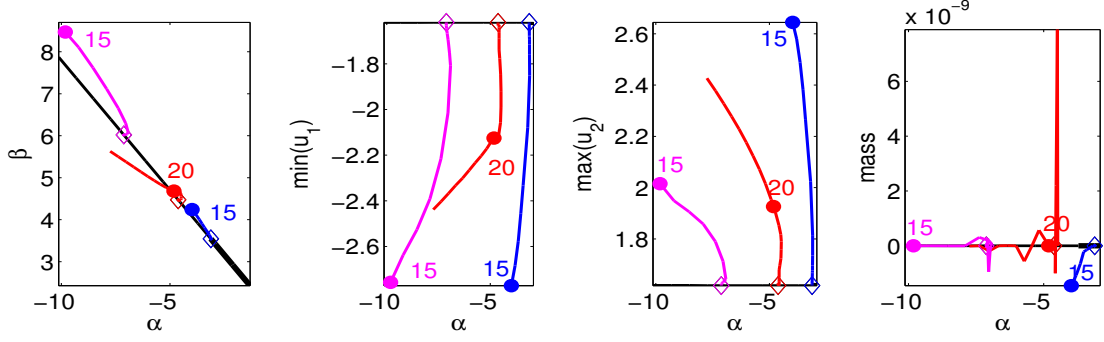
As a Hopf version of (5.4) we use

$$Q_H(u(\cdot, \cdot)) := \sum_{i=1}^m \left( \int_{\Omega} (u_1(t_i, x) + u_2(t_i, x)) \, dx - m \right) \stackrel{!}{=} 0, \quad (5.6)$$

see Listings 19 and 20. In (5.6) i.e., we require the average (in  $t$ ) mass to be conserved. Theoretically it would be sufficient to require  $\int_{\Omega} (u_1(t_0, x) + u_2(t_0, x)) \, dx - m = 0$ , but it turns out that (5.6) is more robust numerically, and that also with (5.6) we have  $\left| \int_{\Omega} (u_1(t_i, x) + u_2(t_i, x)) \, dx - m \right| \leq \text{tol}$  for all  $i$ , i.e., pointwise in  $t$ .

The implementation of (5.5) is rather straightforward, see Table 5 for an overview, and Listings 18–20. We fix  $d_2 = 10$  and restrict to 1D, namely  $\Omega = (-\pi, \pi)$ . Figure 7(b) shows a basic bifurcation diagram, with various quantities as functions of  $\alpha$ . The continuation of (5.5) in  $\alpha$  with fixed  $m = 0$  yields that the homogeneous solution  $u$  stays fixed, i.e.,  $u_1 = -1/2 - \sqrt{5/4}$ ,  $u_2 = -u_1$  for all  $\alpha$ , and that only  $\beta$  is adjusted, see the black lines in (b). (c) shows a number of Hopf orbits, where on each orbit we have  $|Q(u(t, \cdot))| < 10^{-8}$  (see also the last plot in (a) for the average  $Q_H$ ), where the tolerance for the Hopf orbits is  $10^{-6}$ . These Hopf orbits are all unstable according to the associated Floquet multipliers, see also Fig. 8(a), and thus it is interesting to see the evolution of solutions starting on a Hopf orbit (with the numerical error acting as a perturbation of the true point on a Hopf orbit). In Fig. 8(b) we exemplarily show this for the case of  $u(0)$  from `h3/pt15`; here, as in all other cases we considered, the time evolution converges to another stable spatially homogeneous steady state. From this we may again start continuation in, e.g.,  $\alpha$  and  $\beta$ , and find that this branch again typically shows some Hopf bifurcations.

(a) BDs, parameter  $\beta$ ,  $\min(u_1)$ ,  $\max(u_2)$  and mass as functions of  $\alpha$ .



(c) Selected solution plots (both components for h1/pt15)

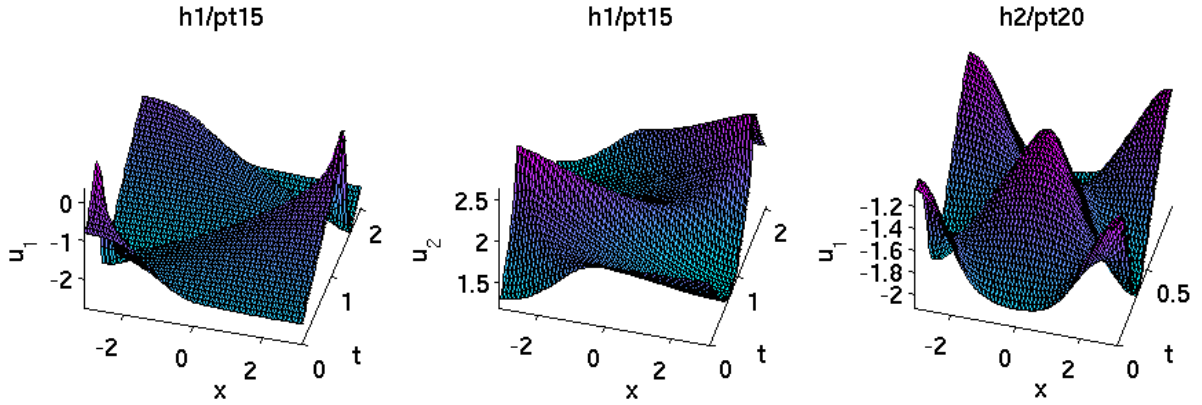


Figure 7: Continuation in  $\alpha$  for (5.5) with  $m = 0$ . (a) Branch data on the homogeneous branch (black) and on three Hopf branches h1 (blue), h2 (red), and h3 (magenta). (b) Example solution plots.

Table 5: Scripts and functions in `hopfdemos/mass-cons`.

script/function	purpose,remarks
<code>cmds1d</code>	main script
<code>mcinit</code> , <code>oosetfemops</code> , <code>sG</code> , <code>sGjac</code> , <code>nodalf</code>	initialization, FEMops, rhs, Jac., and nonlinearity, as usual.
<code>qf</code> , <code>qfjac</code>	the phase condition (5.4), and its Jacobian.
<code>qfh</code> , <code>qfhjac</code>	the Hopf version (5.6) of (5.4), and its Jacobian.

```
function q=qf(p,u) % mass constraint int u1+u2 dx=0
M=p.mat.M(1:p.np,1:p.np); par=u(p.nu+1:end); m=par(4);
u1=u(1:p.np); u2=u(p.np+1:2*p.np); q=sum(M*(u1+u2))/p.vol-m;
```

Listing 18: `mass-cons/qf.m`; mass constraint for steady state computations.

```
function q=qfh(p,y) % aux eqns in Hopf, here: mass constraint
M=p.mat.M(1:p.np,1:p.np); par=p.u(p.nu+1:end); m=par(4); q=0;
for i=1:p.hopf.tl; % sum up masses, i.e., conserve m on average
    u1=y(1:p.np,i); u2=y(p.np+1:2*p.np,i); q=q+sum(M*(u1+u2))/p.vol-m;
end
```

Listing 19: `mass-cons/qfh.m`; Hopf setting of mass constraint. The summing up (in  $t$ ) of the masses turns out to be more robust, with the mass-constraint actually fulfilled pointwise (in  $t$ ).

```
function qjac=qfhjac(p,y) % u-derivatives of qfh
qjac=zeros(1,p.hopf.tl*p.nu); j=p.mat.M(1:p.np,1:p.np)*ones(p.np,1)/p.vol;
for i=1:p.hopf.tl; % same derivative at each time slice
    qjac((i-1)*p.nu+1:i*p.nu)=[j; j]';
end
```

Listing 20: `mass-cons/qfhder.m`,  $u$ -derivatives of  $Q_H$ , cf. last line of (A.10), where the parameter derivatives are done automatically via finite differences.

(a) Floquet multipliers for h1/pt15



(b) Time evolution of small perturbation of  $u$  from h1/pt15 at  $t = 0$ , left  $u_1$ , right  $u_2$

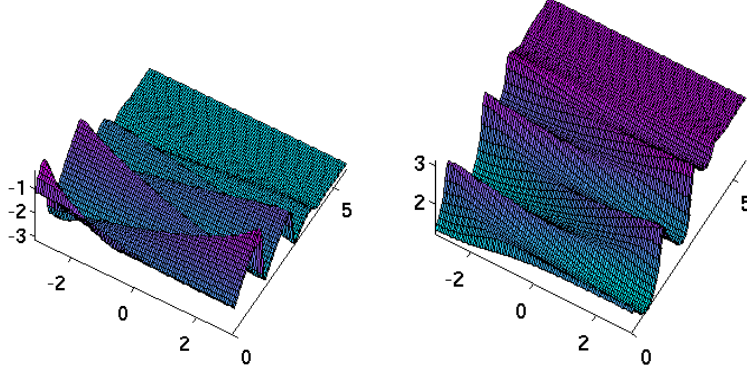


Figure 8: (a) Instability of h1/pt15 as seen in its Floquet multipliers. (b) time integration, with convergence to another spatially homogeneous steady state.

```

close all; format compact; keep pphome;
%% C1: init, and continuation of hom branch
ndim=1; dir='hom1d'; p=[]; lx=pi; nx=100; % domain size and spat.resolution
par=[1; 10; 0; 0; 1; 1]; % d1 d2 d3 m a1 b1
5 p=mcinit(p,lx,nx,par,ndim); p=setfn(p,dir); % initialization
p.nc.nq=1; p.fuha.qf=@qf; % 1 steady constraint (mass), and its func.handle
p.sw.qjac=1; p.fuha.qfder=@qfjac; % use analytical jac for q, and func.handle
p.nc.xiq=0.1; p.nc.ilam=[5 6]; % weight of constr. in arclength, active vars
p=cont(p,20); % run the continuation
10 %% C2: hopf with constraints, passed to hoswibra via aux vars in aux
para=4; ds=0.005; aux=[]; aux.dlam=0; aux.nqnew=0; aux.tl=50;
aux.xif=50; aux.pcfac=10; % weight factors, see hostanparam
aux.nqh=1; aux.qfh=@qfh; aux.qfhder=@qfhjac; % func.handles to hopf constraints
for i=1:3; % continue for 15 steps, first three with large tol
15 p=hoswibra('hom1d',['hpt' mat2str(i)],ds,para,['h' mat2str(i)],aux);
p.nc.ilam=5; p.hopf.ilam=6; p.sw.verb=0; p.hopf.sec=1; p.nc.dsmax=0.5;
p.file.smod=5; p.nc.tol=1e-2; p=cont(p,3); p.nc.tol=1e-4; p=cont(p,12);
end
%% C3: time integrate from some point on Hopf orbit, preparations
20 p=loadp('h2','pt10'); p.u(1:p.nu)=p.hopf.y(1:p.nu,1); % load Hopf point, and
% adjust some settings; in particular store system stiffness matrix in p.mat.K
dir='t1'; u0=p.u(1:p.nu); p=setfn(p,dir);
ts=[]; t0=0; npp=50; nt=200; pmod=50; smod=2; tsmod=1; nc=0;
par=p.u(p.nu+1:end); Ks=p.mat.K; p.mat.K=[[par(1)*Ks par(2)*Ks];[par(3)*Ks Ks]];
25 %% C4: time-integration (repeat if necessary)
[p,t0,ts,nc]=hotintxs(p,u0,t0,ts,npp,nt,nc,tsmod,pmod,smod,@nodalf,1);
%% C5: x-t plot (of both components)
si=0; incr=4; vv=[30,70]; nt=70;
tintplot1d(dir,si,incr,nt,1,1,vv);
30 %% C6: continue from result of tint; again hopf for decreasing alpha
plotsol(p); p.sol.restart=1; p.sol.ds=-0.1; p.sw.para=2; % reset settings to
% steady case, in particular restore scalar stiffness matrix (-Laplacian)

```



```
p.mat.K=Ks; p=rmfield(p,'hopf'); p=setfn(p,'hom1d2'); p=resetc(p);
```

Listing 21: `mass-cons/cmds1d.m`. C1 continues the homogeneous branch, giving a number of Hopf bifurcations; here  $u_1 = -(1 + \sqrt{5})/2$  and  $u_2 = -u_1$  stay fixed, and only  $\beta$  varies with  $\alpha$ . In C2 we follow the first three Hopf branches, where we replace the stationary Hopf constraint in `qf` by the Hopf version `qfh`, see Fig. 7 for bifurcation diagrams and example Hopf solutions. All Hopf branches turn out to be unstable (from the Floquet multipliers), and thus in C3-5 we exemplarily look into the time evolution from the first point ( $t = 0$ ) on the Hopf orbit `h1/pt15`. This converges to a (stable) homogeneous solution again, but at larger amplitude. Finally in C6 we use this as a starting point for continuation in  $\alpha$ , and again find a number of Hopf bifurcations for decreasing  $\alpha$ .

## 5.2 Mass and phase constraints: Demos `kspbc4` and `kspbc2`

The Kuramoto-Sivashinsky (KS) equation [KT76, Siv77] is a canonical and much studies model for long-wave instabilities in dissipative systems, for instance in laminar flame propagation, or for surface instabilities of thin liquid films. Here we consider the KS equation in the form

$$\partial_t u = -\alpha \partial_x^4 u - \partial_x^2 u - \frac{1}{2} \partial_x(u^2), \quad (5.7)$$

with parameter  $\alpha > 0$ , on the 1D domain  $x \in (-2, 2)$  with periodic BC. (5.7) is thus translationally invariant, and has the boost invariance  $u(x, t) \mapsto u(x - ct) + c$ , and we need two phase conditions,

$$\frac{1}{|\Omega|} \int_{\Omega} u \, dx = m, \text{ fixing the mass } m, \quad (5.8a)$$

$$\langle \partial_x u^*, u - u^* \rangle = 0, \text{ fixing the translational invariance.} \quad (5.8b)$$

Here fixing  $m = 0$ , (5.7) shows bifurcations from the trivial solution  $u \equiv 0$  to stationary spatially periodic solutions at  $\alpha_k = \left(\frac{2}{k\pi}\right)^2$ ,  $k \in \mathbb{N}$ . Next, for decreasing  $\alpha$  we obtain secondary Hopf bifurcations from some branches of steady patterns, and for  $\alpha \rightarrow 0$  the dynamics become more and more complicated, making (5.7) a model for turbulence. In [BvVF16], a fairly complete bifurcation diagram (with  $\alpha$  in the range 0.025 to 0.4) has been obtained for (5.7) on  $\Omega = (0, 2)$  with *Dirichlet* BC, i.e.,  $u(0, t) = u(2, t) = \partial_x^2 u(0, t) = \partial_x^2 u(2, t) = 0$ , where in particular many bifurcations have been explained analytically as hidden symmetries by extending solutions antisymmetrically to the domain  $(-2, 2)$  with periodic BC.

Here we directly study (5.7) in this setting, giving us the opportunity to also explain how to setup 4th order equations and periodic BC in `pde2path`. For the latter we only need to call `p=box2per(p,1)`, which generates matrices `fill` and `drop` which are used to transform the FEM matrices such as  $M$  and  $K$  to the periodic setting, see [DU17]. In order to implement 4th order equations there are basically two options:

- (i) Since  $-\partial_x^2 u = M^{-1}Ku$  in the FEM sense, (5.7) can be written in the `pde2path` FEM setting as  $M\partial_t u = -\alpha KM^{-1}Ku + Ku - \frac{1}{2}K_x(u^2)$ . For pBC,  $K, M$  commute, and thus we can multiply by  $M$  to obtain  $M^2\partial_t u = -\alpha K^2u + MKu - \frac{1}{2}MK_x(u^2)$ . Then letting  $M_0=M$  and redefining  $M=M^2$  we obtain  $M\partial_t u = -\alpha K^2u + M_0Ku - \frac{1}{2}M_0K_x(u^2)$ . To incorporate the phase conditions (5.8) we introduce the parameters  $s$  for phase-conservation and  $\varepsilon$  for mass conservation, and thus ultimately consider the system

$$M\dot{u} = -\alpha K^2u + M_0Ku - \frac{1}{2}M_0K_x(u^2) + sK_xu + \varepsilon, \quad (5.9a)$$

$$0 = q_1(u) := \frac{1}{|\Omega|} \sum_{i=1}^{n_u} (M_0u)_i - m, \quad (5.9b)$$

$$0 = q_2(u) := \langle \partial_x u^*, u - u^* \rangle, \quad (5.9c)$$

where  $\frac{1}{|\Omega|} \sum_{i=1}^{n_u} (M_0u)_i$  is the (Riemann sum) approximation of  $\frac{1}{|\Omega|} \int_{\Omega} u \, dx$ , and  $u^*$  is a suitable reference profile. This set up is implemented in `kspbc4`, see below.

- (ii) Alternatively we can rewrite the 4th order equation as a 2 component 2nd order system, for instance for (5.7) in the form

$$\partial_t u = -\alpha \partial_x^2 v - \partial_x^2 u - \frac{1}{2} \partial_x(u^2), \quad 0 = -\partial_x^2 u + v. \quad (5.10)$$

By exploiting the mass matrix on the lhs of (1.3), (5.10) can be straightforwardly implemented in `pde2path` in the form

$$\mathcal{M}\dot{U} = -G(U), \quad U = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \quad \mathcal{M} = \begin{pmatrix} M & 0 \\ 0 & 0 \end{pmatrix}, \quad G(U) = - \begin{pmatrix} K & \alpha K \\ K & M \end{pmatrix} U + \begin{pmatrix} \frac{1}{2} K_x(u_1^2) \\ 0 \end{pmatrix}, \quad (5.11)$$

where  $M, K$  and  $K_x$  are the scalar mass, stiffness and advection matrices. Importantly, the spectral picture and time evolution for (5.11) are still fully equivalent to (5.7). Adding phase conditions like (5.9b,c), this is implemented in `kspbc2`, and yields the same results as the `kspbc4` set up, except for small differences wrt to Floquet multipliers, which in any case are somewhat delicate for constrained Hopf orbits, see Remark A.1.

The implementation of (5.9) is rather straightforward. See Table 6 for an overview, Listings 23–26 for pertinent sections from `oosetfemops`, `cmds1d`, `sG`, `qf` and `qfh`, while for `cmds2.m` and Jacobians/derivatives of `sG`, `qf` and `qfh` we refer to the m-files `sGjac`, `qjac` and `qfhjac`, respectively.

Table 6: Scripts and functions in `hopfdemos/kspbc4`.

script/function	purpose, remarks
<code>cmds1</code>	main script, steady state branches, and associated Hopf bifurcations of standing waves
<code>cmds2</code>	script for one traveling wave branch, and associated Hopf bifurcations of modulated traveling waves
<code>ksinit</code> , <code>oosetfemops</code>	initialization and FEMops; this is somewhat different from the other examples. <code>ksinit</code> also contains the call <code>p=box2per(p,1)</code> to set up the periodic BC; <code>oosetfemops</code> contains calls of <code>filltrafo</code> , and specifically the redefinition of $M$ as $M^2$ .
<code>sG</code> , <code>sGjac</code>	rhs, Jacobian; again somewhat different from before due to 4th order derivatives.
<code>qf</code> , <code>qfjac</code>	the phase conditions (5.9b,c), and the derivatives
<code>qfh</code> , <code>qfhjac</code>	the Hopf version of <code>qf</code> and its derivative

```

%% commands for 1D KS, incl. Hopf branches
close all; keep pphome;
%% C1: init and zero-branch
p=[]; lx=2; nx=100; % domain and discr
5 al=0.42; m=0; par=[al; m; 0; 0]; % m=mass, par(3)=eps, par(4)=s (speed)
p=ksinit(p,nx,lx,par); p=setfn(p,'0'); screenlayout(p);
p.nc.mu1=10; p.nc.mu2=1; % large spacing of evals, be loose about localization
p.nc.ilam=[1 3]; p=cont(p,40); % initial steps
p.sol.ds=-0.001; p.nc.dsmax=0.001; % some more steps with smaller stepsize
10 p.nc.mu2=5; p=cont(p,20); p.file.smod=10; p=cont(p,10);
%% C2: compute branches of steady patterns
for i=1:4
    is=mat2str(i); p=swibra('0',['bpt' is],is,i*0.01);
    p.file.smod=20; p.sw.bifcheck=0; p=cont(p,5); % a few steps without PC
15 p.u0x=p.mat.Kx*p.u(1:p.nu); % set profile for transl-invariance
    p.nc.nq=2; p.nc.ilam=[1 3 4]; p.tau=[p.tau; 0]; % now switch on PCs
    p.sw.bifcheck=2; p.nc.dsmax=0.2; p.nc.tol=1e-6; p=cont(p,i*80);
end
%% C3: 1st Hopf bifurcation
20 figure(2); clf; ds=0.1; clear aux; aux.dlam=0; aux.nqnew=0; aux.tl=30;
aux.xif=0.1; aux.y0dsw=2; % use PDE to set d/dt u_0 for phase-constr. (in t)
aux.nqh=2; aux.qfh=@qfh; aux.qfhder=@qfhjac; % func handles to hopf constraints
p=hoswibra('2','hpt1',ds,4,'h1',aux); p.nc.ilam=1; p.hopf.ilam=[3 4];
p.hopf.fltol=1e-2; p.hopf.nfloq=10; p.hopf.flcheck=2; p.sw.verb=0;

```



```

25 p.hopf.sec=1; p.nc.tol=1e-6; p.nc.dsmax=0.3; p.file.smod=2; p=cont(p,1);
   p.hopf.flcheck=1; p=cont(p,14); % floqps fails for larger amplitudes,
   % hence switch to floq: caution, only large multipliers seem correct

```

Listing 22: `kspbc4/cmds1.m`. Cell 1 deals with initialization and continuation of the trivial branch. Since the eigenvalues  $\mu_k = -\alpha(k\pi/2)^4 + (k\pi/2)^2$  of the linearization around  $u \equiv 0$  have a rather large spacing, in line 7 we set  $\mu_{1,2}$  (see (A.2)) to rather large values. In Cell 2 we compute the first 4 branches of steady patterns. The phase condition  $\langle \partial_x u^*, u \rangle = 0$  (2nd component of `qf`, see Listing 25) is only switched on after a few initial steps and then setting the reference profile  $\partial_x u^* = p.u0x$ , because it only makes sense for  $u^*$  not spatially homogeneous. C3 computes the first Hopf branch `h1`, bifurcating from steady branch 2. We use a rather large Floquet tolerance `p.hopf.fltol`, see (2.5), because the Floquet computations do not remove the neutral directions, cf. Remark A.1. Moreover, for this problem `floqps` for the multiplier computations via periodic Schur decomposition sometimes fails (for unknown reasons), while `floq` (for unknown but maybe related reasons) seems somewhat unreliable for the small multipliers; the large multipliers (and hence the stability information) however always seem correct. The remainder of `kspbc4/cmds1.m` deals with the Hopf branches `h2` and `h3`, and with plotting.

```

function p=oosetfemops(p) % with filltrafo to transform to per.domain
gr=p.pdeo.grid;
[K,M,~]=p.pdeo.fem.assema(gr,1,1,1); Kx=convection(p.pdeo.fem,gr,1);
p.mat.K=filltrafo(p,K); M=filltrafo(p,M); p.mat.Kx=filltrafo(p,Kx);
5 p.mat.M0=M; p.mat.M=M^2; % save M as M0 and redefine M for the 4th order setup

```

Listing 23: `kspbc4/oosetfemops.m`.

```

function r=sG(p,u) % KS in 4th order formulation
K=p.mat.K; M0=p.mat.M0; Kx=p.mat.Kx; par=u(p.nu+1:end);
al=par(1); eps=par(3); s=par(4); u=u(1:p.nu); uxx=K*u;
r=al*K*uxx-M0*uxx+0.5*M0*(Kx*(u.^2))+s*M0*(Kx*u)+eps;

```

Listing 24: `kspbc4/sG.m`. Note that the mass matrix `p.mat.M` has been redefined in `oosetfemops` to  $M^2$ , and the proper mass matrix is stored in `p.mat.M0`.

```

function q=qf(p,u) % mass (and phase) constraint for KS
par=u(p.nu+1:end); u=u(1:p.nu); % extract pars and u-vars
q=sum(p.mat.M0*u)/p.vol-par(2); % mass constraint
if p.nc.nq==2; % if active, then add phase constraint
5   if isfield(p,'u0x'); u0x=p.u0x(1:p.nu); else u0x=p.mat.Kx*p.u(1:p.nu); end
   q=[q;u0x'*u];
end

```

Listing 25: `kspbc4/qf.m`. The two phase conditions for the steady case.

```

function q=qfh(p,y) % aux eqns in Hopf, here: sum up shifts wrt u0
par=p.u(p.nu+1:end); m=par(2); n=p.nu;
q1=sum(p.mat.M0*y(1:n,1))/p.vol-m; % mass constraint (at initial slice)
tl=size(p.hopf.y,2); q2=0; % phase constr, useful to define 'on average'
5 if isfield(p,'u0x'); u0x=p.u0x(1:p.nu); else u0x=p.mat.Kx*p.u(1:p.nu); end
for i=1:tl; u=y(1:n,i); q2=q2+u0x'*u; end
q=[q1;q2];

```

Listing 26: `kspbc4/qfh.m`. The Hopf setting of the two phase conditions.

Figure 9(a) shows a basic bifurcation diagram of steady states, including one branch of traveling waves, obtained from `cmds2.m`. As predicted, at  $\alpha_k$  we find supercritical pitchforks of steady branches. The first one starts out stable, and loses stability in another supercritical pitchfork around  $\alpha = 0.13$  to a traveling wave branch (brown), which then loses stability in a Hopf bifurcation, see Fig. 11. However, here we first focus on Hopf bifurcations from the 2nd and 3rd primary branches, which first gain stability at some rather large amplitude, then lose it again in Hopf bifurcations, with the solution profiles at the HBP in (c). (b) zooms into the BD at low  $\alpha$ , including the 4th steady branch, and three Hopf branches, while (d) shows selected Hopf orbits, and Fig. 10 associated multiplier spectra.

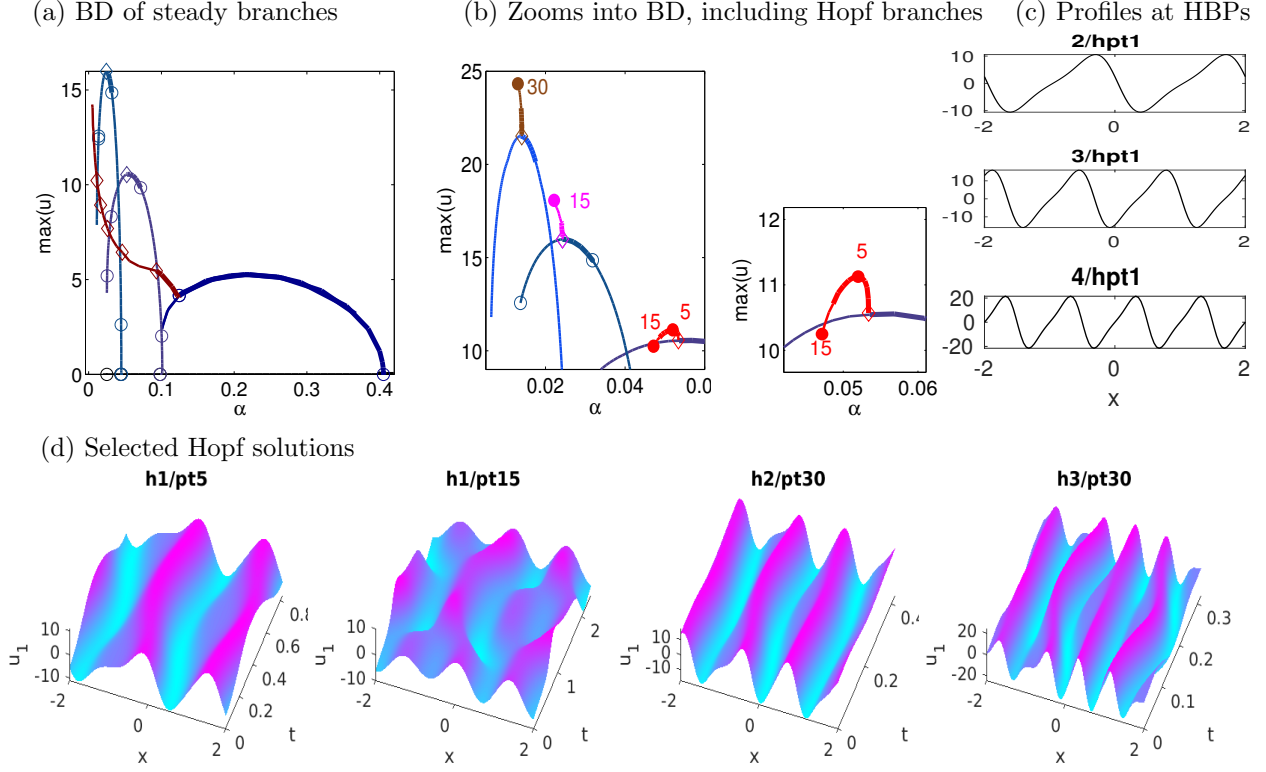


Figure 9: Results from `kspbc4/cmds1.m`. Bifurcation diagrams of steady solutions (except for the brown branch of traveling waves, see Fig. 11) (a), with zoom in (b), including the 4th steady branch and 3 Hopf branches  $h1$  (red),  $h2$  (magenta) and  $h3$  (brown). For all these branches  $m = \varepsilon = 0$  (numerically  $\mathcal{O}(10^{-10})$ ), and except for the brown branch in (a) also  $s = 0$ . Profiles at the Hopf bifurcation points the steady branches in (c). In (d) we plot selected Hopf orbits and multiplier spectra.  $h1$  loses stability at  $\alpha \approx 0.0486$  in a pitchfork (a multiplier becoming unstable at  $\mu = 1$ ), and the largest multiplier of  $h1/pt15$  is  $\mu_2 \approx 4000$ . Also  $h2$  is initially stable, but loses stability in a pitchfork at  $\alpha \approx 0.024$ , i.e., rather close to bifurcation, and a similar behaviour occurs on  $h3$ . See Fig. 10 for multiplier plots, and `cmds2.m` and Fig. 11 for further plots, for instance of solutions on the secondary brown branch in (a), and the Hopf bifurcations from this branch.

These results all fully agree with those in [BvVF16] (by extending the solutions from [BvVF16] antisymmetrically), who however proceed further by also computing some (standing) Hopf branches bifurcating in pitchforks and period doublings from the above (standing) Hopf branches. While these bifurcations are also detected in `pde2path`, the branch switching is not yet implemented. On the other hand, for our periodic BC on the larger domain we also have traveling waves and Hopf bifurcations to modulated traveling waves. Some examples for these are considered in `cmds2.m`, see also Fig. 11.

## A A collection of formulas

For details and background on the algorithms we refer to [Uec17a], but here briefly repeat the pertinent formulas that are implemented in `pde2path`, including the augmented systems for Hopf computations with constraints. We focus on the arclength parametrization setting (`p.sw.param=4`), which is more convenient and robust than the 'natural parametrization' (`p.sw.param=3`).

First of all, the detection of Hopf bifurcation points (HBPs) requires the `p.sw.bifcheck=2`

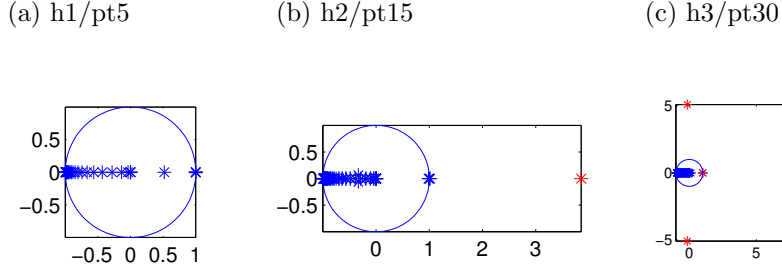


Figure 10: Multiplier spectra for the orbits from Fig. 9(d).

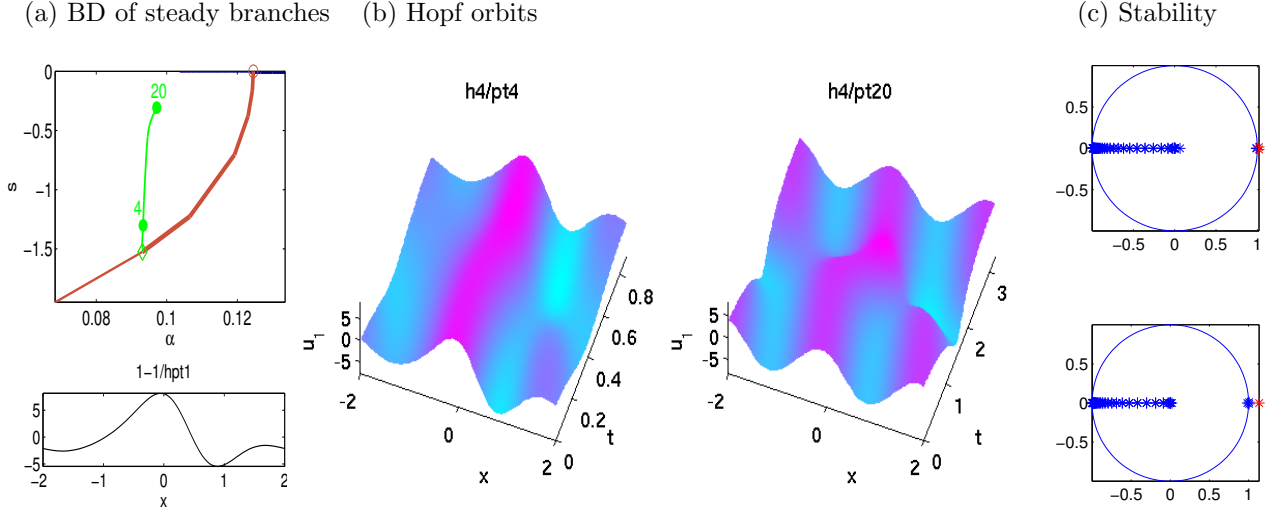


Figure 11: Results from `kspbc4/cmds2.m`. (a) Bifurcation diagram ( $s$  over  $\alpha$ ) of the traveling wave branch from Fig. 9(a), and of the first bifurcating modulated traveling wave branch (green), with the solution profile at bifurcation at the bottom. (b) shows two Hopf orbits on the green branch (in the frames moving with speeds  $s$  from (a), respectively), and (c) the associated Floquet multipliers. The bifurcation is subcritically, and the modulated traveling waves are (mildly) unstable.

setting [Uec17a, §2.1], which is essentially controlled as follows:

`p.nc.eigref(1:ne)` contains shifts near which eigenvalues are computed; guesses for these shifts can be obtained via `initedig`. However, except for §(3.2), here we use `p.nc.eigref=0.`) (A.1)

A bisection for localization of a possible HBP is started if  $|\operatorname{Re}\mu| < \mathbf{p.nc.mu1}$  for the eigenvalue with the smallest abs. real part, and a HBP is accepted if  $|\operatorname{Re}\mu| < \mathbf{p.nc.mu2}$  at the end of the bisection. (A.2)

The default branch switching `hoswibra` to a Hopf branch generates

$$\lambda = \lambda_H + \delta_s \delta_\lambda \quad u(t) = u_0 + 2\alpha \delta_s \Re(e^{-i\omega_H t} \Psi), \quad (\text{A.3})$$

as an initial guess for a periodic solution of (1.3) with period near  $2\pi/\omega$ . Here  $\Psi$  is the (complex) eigenvector associated to  $i\omega_H$ , and  $\delta_\lambda, \alpha$  are computed from the normal form

$$0 = r \left[ \mu'_r(\lambda_H)(\lambda - \lambda_H) + c_1 |r|^2 \right]. \quad (\text{A.4})$$

of the bifurcation equation on the center manifold, see [Uec17a, §2.2]. After the coefficients  $\delta_\lambda$  and  $\alpha$  in (A.3) are computed (in `hogetnf`),  $\delta_s$  is chosen in such a way that the initial step length is  $ds$  in the norm (A.8) below.

This so far applies to semilinear systems, i.e., in FEM form  $M\dot{u} = Ku - Mf(u)$  where the stiffness matrix  $K$  does not depend on  $u$ . For more general problems, or in general as an alternative to the computation of  $\delta_\lambda$  and  $\alpha$ , the user can also pass values for  $\delta_\lambda$  and  $\alpha$  to `hoswibra`. Often,  $\delta_\lambda = 0$  (and then automatically  $\alpha = 1$  by normalization) works fine.

To compute Hopf orbits, after rescaling  $t \mapsto Tt$  with unknown period  $T$ , the time evolution and periodicity condition for  $u$  read

$$M\dot{u} = -TG(u, \lambda), \quad u(\cdot, 0) = u(\cdot, 1), \quad (\text{A.5})$$

and the time-translational phase condition and arclength equation read

$$\phi := \xi_\phi \int_0^1 \langle u(t), \dot{u}_0(t) \rangle dt \stackrel{!}{=} 0, \quad (\text{A.6})$$

$$\psi := \xi_H \sum_{j=1}^m \langle u(t_j) - u_0(t_j), u'_0(t_j) \rangle_\Omega + (1 - \xi_H) [w_T(T - T_0)T'_0 + (1 - w_T)(\lambda - \lambda_0)\lambda'_0] - ds \stackrel{!}{=} 0, \quad (\text{A.7})$$

where  $T_0, \lambda_0$  and  $u_0$  are from the previous step,  $\xi_H, w_T$  are weights,  $'$  denotes differentiation wrt arclength, and  $\langle u, v \rangle_\Omega$  stands for  $\int_\Omega \langle u(x), v(x) \rangle dx$ , with  $\langle a, b \rangle$  the standard  $\mathbb{R}^N$  scalar product. Numerically, we use  $\langle u, v \rangle_\Omega = \langle Mu, v \rangle$ , where  $M$  is the mass matrix belonging to the FEM mesh. The steplength is  $ds$  in the weighted norm

$$\|(u, T, \lambda)\|_\xi = \sqrt{\xi_H \left( \sum_{j=1}^m \|u(t_j)\|_2^2 \right) + (1 - \xi_H) [w_T T^2 + (1 - w_T) \lambda^2]}. \quad (\text{A.8})$$

In (A.6),  $\xi_\phi$  with standard setting  $\xi_\phi = 10$  (`p.hopf.pcfac`, see Appendix B) is another weight, which can be helpful to balance the Jacobian  $\mathcal{A}$ , see (A.10). To improve convergence of Newton loops, it sometimes turns out to be usefull to set  $\xi_\phi$  to somewhat larger values. Also, while  $\dot{u}_0$  in (A.6) is in principle available from  $M\dot{u}_0 = -TG(u_0, \lambda_0)$  (which is used for `p.hopf.y0dsw=0`, it often appears more robust to explicitly approximate  $\dot{u}_0$  via finite differences, for which we set `p.hopf.y0dsw=2`. Finally, for a system with  $n_H$  Hopf constraints  $Q_H(u) = 0$ , and hence  $n_H$  additional free parameters  $a \in \mathbb{R}^{n_H}$ , we also add  $w_a \langle a - a_0, a' \rangle$  to  $\psi$ , where  $w_a$  is a weight for the auxiliary parameters  $a$ .

Letting  $U = (u, T, \lambda, a)$ , and writing  $\mathcal{G}(U) = 0$  for (A.5) (see also (A.12)), in each continuation step we need to solve

$$H(U) := \begin{pmatrix} \mathcal{G}(U) \\ \phi(u) \\ \psi(U) \\ Q_H(U) \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \in \mathbb{R}^{mn_u + 2 + n_H}, \quad (\text{A.9})$$

where  $m$  is the number of time slices  $t_j$ ,  $j = 1, \dots, m$ ,  $n_u$  the number of PDE unkowns at each time slice, and  $n_H =: \text{p.hopf.nqh}$  is the number of Hopf constraints (encoded in `p.hopf.qfh`). To solve (A.9) we use Newton's method, i.e.,

$$U^{j+1} = U^j - \mathcal{A}(U^j)^{-1} H(U^j), \quad \mathcal{A} = \begin{pmatrix} \partial_u \mathcal{G} & \partial_T \mathcal{G} & \partial_\lambda \mathcal{G} & \partial_a \mathcal{G} \\ \partial_u \phi & 0 & 0 & 0 \\ \xi_H \tau_u & (1 - \xi_H) w_T \tau_T & (1 - \xi_H)(1 - w_T) \tau_\lambda & w_a \tau_a \\ \partial_u Q_H & \partial_T Q_H & \partial_\lambda Q & \partial_a Q_H \end{pmatrix}, \quad (\text{A.10})$$

where of course we never form  $\mathcal{A}^{-1}$  but instead use `p.fuha.blss` to solve linear systems of type  $\mathcal{A}U = b$ . These systems are of bordered type, and thus it is sometimes advantageous to use bordered system solvers, see [UW17b].

For the time discretization we have

$$u = (u_1, \dots, u_m) = (u(t_1), u(t_2), \dots, u(t_m)), \quad (\text{A.11})$$

( $m$  time slices, stored in `p.hopf.y(1:p.nu, 1:m)`),

and to assemble  $\mathcal{G}$  in (A.5) we use modifications of TOM, yielding, with  $h_j = t_j - t_{j-1}$  and  $u_0 := u_{m-1}$ ,

$$(\mathcal{G}(u))_j = -h_{j-1}^{-1}M(u_j - u_{j-1}) - \frac{1}{2}T(G(u_j) + G(u_{j-1})), \quad \mathcal{G}_m(u) = u_m - u_1. \quad (\text{A.12})$$

The Jacobian is  $\partial_u \mathcal{G} = A_1$ , where we set, as it is also used for the Floquet multipliers with free  $\gamma$  (see [Uec17a, §2.4]),

$$A_\gamma = \begin{pmatrix} M_1 & 0 & 0 & 0 & \dots & -H_1 & 0 \\ -H_2 & M_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & -H_3 & M_3 & 0 & \dots & 0 & 0 \\ \vdots & \dots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \dots & \dots & \ddots & \ddots & 0 & 0 \\ 0 & \dots & \dots & 0 & -H_{m-1} & M_{m-1} & 0 \\ -\gamma I & 0 & \dots & \dots & \dots & 0 & I \end{pmatrix}, \quad (\text{A.13})$$

where  $M_j = -h_{j-1}^{-1}M - \frac{1}{2}T\partial_u G(u_j)$ ,  $H_j = -h_{j-1}^{-1}M + \frac{1}{2}T\partial_u G(u_{j-1})$ , and  $I$  is the  $n_u \times n_u$  identity matrix.<sup>6</sup> The Jacobians  $\partial_u G \in \mathbb{R}^{n_u \times n_u}$  in  $M_j, H_j$  are computed as for steady state problems, e.g., via `p.fuha.sGjac`, or by `numjac` if `p.sw.jac=0` (but still locally in time).

For  $Q_H(u(\cdot, \cdot), \lambda, w, a) = 0$ , the user must provide a function handle `p.hopf.qfh`, similar to  $0 = Q(u, \lambda, w) = \text{p.fuha.qf}$  for the steady case. Moreover, when switching to a Hopf branch (and if `p.nc.nq` was greater 0 for the steady continuation), the user has to drop the stationary constraints, i.e., reset `p.nc.nq=0` and `p.nc.ilam=p.nc.ilam(1)` to just the primary active parameter, while the other active parameters  $a \in \mathbb{R}^{n_H}$  for  $Q_H$  should be set in `p.hopf.ilam`  $\in \mathbb{N}^{n_H}$ , which now acts as a pointer to these 'secondary' active parameters. Finally, the user must provide a function handle in `p.hopf.qfhjac` to a function that returns  $\partial_u Q_H$  from the last line in (A.10). On the other hand,  $\partial_T \mathcal{G}, \partial_\lambda \mathcal{G}, \partial_a \mathcal{G}$ , and  $\partial_T Q_H, \partial_\lambda Q_H$  and  $\partial_a Q_H$  in (A.10) are cheap from numerical differentiation and hence taken care of automatically.

**Remark A.1.** (a) Also for  $n_H > 0$ , the computation of Floquet multipliers is based on (A.13), i.e., ignores the Hopf-constraints  $Q_H$ . Since these constraints are typically used to eliminate neutral directions, ignoring these typically leads to Floquet multipliers close to 1, additional to the trivial multiplier 1 from (time-) translational invariance. This may lead to wrong stability assessments of periodic orbits (see [RU17a, §4] for an example), which however usually can be identified by suitable inspection of the multipliers.

(b) If (A.5) contains algebraic constraint components as in `kspbc2`, then we modify (A.12). For instance, if the second component is algebraic, then we (automatically, in `tomassemF` and `tomassembc`) set  $(\mathcal{G}(u)_2)_j = -\frac{1}{2}TG(u_j)$ , and accordingly also modify (A.13). ]

---

<sup>6</sup>In (A.12) and (A.13) we assume that  $M$  is regular, i.e., (A.5) does not contain algebraic constraints as for instance for the 2nd order system formulation of the KS in `kspbc2`; see Remark A.1(b) for this case.

## B Data structure and function overview

The Hopf setting naturally reuses and extends the stationary `pde2path` setting explained in, e.g., [dWDR<sup>+</sup>17]. As usual, here we assume that the problem is described by the struct `p`, and for convenience list the main subfields of `p` in Table 7.

Table 7: Main fields in the structure `p` for steady problems, see [dWDR<sup>+</sup>17] for more details.

field	purpose	field	purpose
<code>fuha</code>	<b>f</b> unction <b>h</b> andles, e.g., <code>fuha.G</code> , ...	<code>nc</code>	<b>n</b> umerical <b>c</b> ontrols, e.g., <code>nc.tol</code> , ...
<code>sw</code>	<b>s</b> witches such as <code>sw.bifcheck</code> , ...	<code>sol</code>	values/fields calculated at runtime
<code>pdeo</code>	OOPDE data if OOPDE is used	<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>bel</code>	controls for <code>lssbel</code> (bordered elimination)	<code>ilup</code>	controls for <code>lssAMG</code> (ilupack parameters)
<code>usrlam</code>	vector of user set target values for the primary parameter, default <code>usrlam=[]</code> ;		
<code>mat</code>	problem matrices, in general data that is not saved to file		

For the continuation of time-periodic orbits, the field `p.hopf` contains the pertinent data; it is typically created and filled by calling `p=hoswibra(...)`. This inter alia calls `p=hostanparam(p,aux)`, which can be used as a reference for the default values of the Hopf parameters. The unconstrained Hopf setting does not need any user setup additional to the functions such as `p.fuha.sG`, `p.fuha.sGjac` already needed for stationary problems. In case of Hopf constraints, the user has to provide two function handles in `p.hopf.qfh` and `p.hopf.qfhder` to functions which compute  $Q_H$  from (A.9) and the last row of  $\mathcal{A}$  from (A.10), respectively. 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 8 gives an overview of `p.hopf`, and Table 9 lists the main Hopf orbit related functions.

Table 8: Standard (and additional, at bottom) entries in `p.hopf`.

field	purpose
<code>y</code>	for <code>p.sw.param=4</code> : 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);
<code>y0d</code>	for <code>p.sw.param=3</code> : $y$ augmented by $\tilde{y}$ and $T, \lambda$ ( $(2n_u+2) \times m$ matrix), see [Uec17a].
	for <code>p.sw.param=4</code> : $M\dot{u}_0$ for the phase condition (A.6), ( $n_u \times m$ matrix);
<code>y0dsw</code>	for <code>p.sw.param=3</code> : $M\dot{u}_0(0)$ for the phase condition [Uec17a, (36)], ( $2n_u+2$ vector).
	(for <code>p.sw.param=4</code> ) controls how $\dot{u}_0$ in (A.6) is computed: 0 for using the PDE (A.5), 2 for using FD (default).
<code>pcfac</code>	weight for the phase condition (A.6), default=10
<code>tau</code>	tangent, for <code>p.sw.param=4</code> , $(mn_u + 2 + n_H) \times 1$ vector, see third line in (A.10)
<code>ysec</code>	for <code>p.sw.param=3</code> , secant between two solutions $(y_0, T_0, \lambda_0)$ , $(y_1, T_1, \lambda_1)$ , $(2n_u+2) \times m$ matrix
<code>sec</code>	if <code>sec=1</code> , then use secant tau (instead of tangent) predictor for <code>p.sw.param=4</code>
<code>t, T, lam</code>	time discretization vector, current period and param.value
<code>xi,tw,qw</code>	weights for the arclength (A.7), <code>xi</code> = $\xi_H$ , <code>tw</code> = $w_T$ , <code>qw</code> = $w_a$ ;
<code>x0i</code>	index for plotting $t \mapsto u(\vec{x}(x0i))$ ;
<code>plot</code>	aux. vars to control hoplot during <code>hocont</code> ; see the description of <code>hoplot</code> ; default <code>plot=[]</code>
<code>wn</code>	struct containing the winding number related settings for <code>initeig</code>
<code>tom</code>	struct containing TOM settings, including the mass matrix $M$
<code>jac</code>	switch to control assembly of $\partial_u \mathcal{G}$ . <code>jac=0</code> : numerically (only recommended for testing); <code>jac=1</code> : via <code>hosjac</code> . Note that for <code>p.sw.jac=0</code> the local matrices $\partial_u G(u(t_j))$ are obtained via <code>numjac</code> , but this is still much faster than using <code>p.hopf.jac=0</code> .
<code>flcheck</code>	0 to switch off multiplier-comp. during <code>cont.</code> , 1 to use <code>floq</code> , 2 to use <code>floqps</code>
<code>nfloq</code>	# of multipliers (of largest modulus) to compute (if <code>flcheck=1</code> )
<code>ftol</code>	tolerance for multiplier $\gamma_1$ (give warning if $ \gamma_1 - 1  > \text{p.hopf.ftol}$ )
<code>muv1,muv2</code>	vectors of stable and unstable multipliers, respectively
<code>pcheck</code>	if 1, then compute residual in <code>hoswibra</code> (predictor check)

	Additional entries in case of Hopf constraints
ilam	pointer to the $n_Q$ additional active parameters in <code>p.u(p.nu+1:end)</code> ; the pointer to the primary active parameter is still in <code>p.nc.ilam(1)</code> .
qfh, qfhder	(handles to) functions returning the Hopf constraints $Q_H(U)$ and the derivatives (last lines of $H$ in (A.9) and $\mathcal{A}$ in (A.10), respectively).

Table 9: Overview of main functions related to Hopf bifurcations and periodic orbits; see `p2phelp` for argument lists and more comments.

name	purpose, remarks
hoswibra	branch switching at Hopf bifurcation point, see comments below
hoplot	plot the data contained in <code>hopf.y</code> . 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 <code>hoplotf</code> ;
initeig	find guess for $\omega_1$ ; see also <code>initwn</code>
floq	compute <code>p.hopf.nfloq</code> multipliers during continuation ( <code>p.hopf.flcheck=1</code> )
floqps	use periodic Schur to compute (all) multipliers during continuation ( <code>flcheck=2</code> )
floqap, floqpsap	a posteriori versions of <code>floq</code> and <code>floqps</code> , respectively
hobra	standard-setting for <code>p.fuha.outfu</code> (data on branch), template for adaption to a given problem
hostanufu	standard function called after each continuation step
plotfloq	plot previously computed multipliers
hotintxs	time integrate (1.3) from the data contained in <code>p.hopf</code> and <code>u0</code> , with output of $\ u(t) - u_0\ _\infty$ , and saving $u(t)$ to disk at specified values
tintplot*d	plot output of <code>hotintxs</code> ; $x-t$ -plots for $*$ =1, else snapshots at specified times
hopftref	refine the $t$ -mesh in the arclength setting at user specified time $t^*$
hogradinf	convenience function returning the time $t^*$ where $\ \partial_t u(\cdot, t)\ _\infty$ is maximal; may be useful for <code>hopftref</code> .
initwn	init vectors for computation of initial guess for spectral shifts $\omega_j$
hogetnf	compute initial guesses for <code>diam</code> , <code>al</code> from the normal form coefficients of bifurcating Hopf branches, see [Uec17a, (16)]
hocont	main continuation routine; called by <code>cont</code> if <code>p.sol.ptype&gt;2</code>
hostanparam	set standard parameters
hostanopt	set standard options for hopf computations
hoinistep	perform 2 initial steps and compute secant, used if <code>p.sw.param=3</code>
honloopext, honloop	the arclength Newton loop, and the Newton loop with fixed $\lambda$
sety0dot	compute $\dot{u}_0$ for the phase condition (A.6)
tomsol	use TOM to compute periodic orbit in <code>p.sw.param=3</code> setting.
tomassemG	use TOM to assemble $\mathcal{G}$ , see [Uec17a, (26)]; see also <code>tomassem</code> , <code>tomassembc</code>
gethoA	put together the extended Jacobian $\mathcal{A}$ from [Uec17a, (27)]
hopc	the phase condition $\phi$ from [Uec17a, (19)], and $\partial_u \phi$ .
arc2tom, tom2arc	convert arclength data to <code>tomsol</code> data, e.g., to call <code>tomsol</code> for mesh adaptation. <code>tom2arc</code> to go back.
ulamcheckho	check for and compute solutions at user specified values in <code>p.usrlam</code>
hosrhs, hosrhsjac	interfaces to <code>p.fuha.G</code> and <code>p.fuha.Gjac</code> at fixed $t$ , internal functions called by <code>tomassembc</code> , together with <code>hodummybc</code>
horhs, hojac	similar to <code>hosrhs</code> , <code>horhsjac</code> , for <code>p.sw.param=3</code> , see also <code>hobc</code> and <code>hobcjac</code>

Besides `cont`, the functions `initeig`, `hoswibra`, `hoplot`, `floqap`, `floqpsap`, `floqplot`, `hotintxs`, `tintplot*d`, `hogradinf` and `hopftref` are most likely to be called directly by the user, and `hobra` (the branch data) and `hostanufu` (called after each continuation step) are likely to be adapted by the user. As usual, all functions in Table 9 can be most easily overloaded by copying them to the given problem directory and modifying them there.

In `p=hoswibra(dir, fname, ds, para, varargin)`, the auxiliary argument `aux=varargin{2}` (`varargin{1}` is the new directory) can for instance have the following fields:

- `aux.tl=21`: number of (equally spaced) initial mesh-points in  $t \in [0, 1]$  (might be adaptively

refined by TOM for `p.sw.param=3`, or via `hopftref` or `uhopftref` for `p.sw.param=4`).

- `aux.hodel=1e-4`: used for the finite differences in `hogetnf`.
- `aux.al`, `aux.dlam` (no preset): these can be used to pass a guess for  $\alpha$  and  $\delta_\lambda$  in (A.3) and thus circumvent `hogetnf`; useful for quasilinear problems and for problems with constraints (for which `hogetnf` will not work), or more generally when the computation of  $\alpha, \delta_\lambda$  via `hogetnf` seems to give unreliable results.
- `aux.z`: The coefficients  $z_1, z_2$  in the ad hoc modification (3.5) of (A.3) used for the double Hopf case in demo `rot`.

For the other functions listed above we refer to the m-files for description of their arguments, and to the demo directories for examples of usage and customization.

## References

- [Bol16] U. Bollhöfer. ILUPACK V2.4. <http://www.icm.tu-bs.de/~bolle/ilupack/>, 2016.
- [BPS01] W.J. Beyn, Th. Pampel, and W. Semmler. Dynamic optimization and Skiba sets in economic examples. *Optimal Control Applications and Methods*, 22(5–6):251–280, 2001.
- [BvVF16] P.-L. Buono, L. van Veen, and E. Frawley. Hidden symmetry in a Kuramoto-Sivashinsky initial-boundary value problem, 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.
- [dWDR<sup>+</sup>17] H. de Witt, T. Dohnal, J. Rademacher, H. Uecker, and D. Wetzel. `pde2path` - Quickstart guide and reference card, 2017.
- [GCF<sup>+</sup>08] D. Grass, J.P. Caulkins, G. Feichtinger, G. Tragler, and D.A. Behrens. *Optimal Control of Nonlinear Processes: With Applications in Drugs, Corruption, and Terror*. Springer Verlag, 2008.
- [GKS00] M. Golubitsky, E. Knobloch, and I. Stewart. Target patterns and spirals in planar reaction-diffusion systems. *J. Nonlinear Sci.*, 10(3):333–354, 2000.
- [GS02] M. Golubitsky and I. Stewart. *The symmetry perspective*. Birkhäuser Verlag, Basel, 2002.
- [GU17] D. Grass and H. Uecker. Optimal management and spatial patterns in a distributed shallow lake model. *EJDE*, 2017.
- [Hoy06] R.B. Hoyle. *Pattern formation*. Cambridge University Press., 2006.
- [Kno94] E. Knobloch. Bifurcations in rotating systems. In *Lectures on Solar and Planetary Dynamos, Publications of the Newton Institute, Eds. M.R.E. Proctor and A.D. Gilbert*, pages 247–253. Cambridge University Press, 1994.
- [KT76] Y. Kuramoto and T. Tsuzuki. Persistent propagation of concentration waves in dissipative media far from thermal equilibrium. *Prog. Theoret. Phys.*, 55(2):356–369, 1976.
- [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](http://www.mathe.tu-freiberg.de/nmo/mitarbeiter/uwe-pruefert/software), 2016.
- [RU17a] J. Rademacher and H. Uecker. Symmetries, freezing, and Hopf bifurcations of modulated traveling waves in `pde2path`, 2017.
- [RU17b] J. Rademacher and H. Uecker. The OOPDE setting of `pde2path` – a tutorial via some Allen-Cahn models, 2017.
- [Siv77] G. Sivashinsky. Nonlinear analysis of hydrodynamic instability in laminar flames. I - Derivation of basic equations. *Acta Astronautica*, 4:1177–1206, 1977.
- [Uec16] H. Uecker. Optimal harvesting and spatial patterns in a semi arid vegetation system. *Natural Resource Modelling*, 29(2):229–258, 2016.



- [Uec17a] H. Uecker. Hopf bifurcation and time periodic orbits with pde2path – algorithms and applications, Preprint, 2017.
- [Uec17b] H. Uecker. Infinite time-horizon spatially distributed optimal control problems with pde2path – a tutorial, 2017.
- [Uec17c] H. Uecker. [www.staff.uni-oldenburg.de/hannes.uecker/pde2path](http://www.staff.uni-oldenburg.de/hannes.uecker/pde2path), 2017.
- [UW17a] H. Uecker and D. Wetzel. The pde2path linear system solvers – a tutorial, 2017. Available at [Uec17c].
- [UW17b] 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.
- [Wir00] Fr. Wirl. Optimal accumulation of pollution: Existence of limit cycles for the social optimum and the competitive equilibrium. *Journal of Economic Dynamics and Control*, 24(2):297–306, 2000.
- [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.