Symmetries, freezing, and Hopf bifurcations of traveling waves in `pde2path`

Jens D.M. Rademacher[1], Hannes Uecker[2]

[1] Fachbereich Mathematik, Universität Bremen, D28359 Bremen, jdmr@uni-bremen.de
[2] Institut für Mathematik, Universität Oldenburg, D26111 Oldenburg, hannes.uecker@uni-oldenburg.de

August 16, 2017

Abstract

We use four 1D model problems to explain the setup of phase conditions to handle continuous symmetries in `pde2path`. The first is a complex Ginzburg-Landau equation with, inter alia, translational and rotational invariance. The second is a FitzHugh-Nagumo type system, for which we also implement a 'freezing' method to obtain traveling waves and their speed from time integration. Additionally we describe setups to compute branches of relative periodic orbits, namely modulated fronts for a model of autocatalysis, and breathing pulses for another FHN model.

Contents

1 Introduction

The purpose of this tutorial is to explain the handling of continuous symmetries of 1D problems

$$\partial_t u = -G(u,\lambda), \quad u = u(t,x) \in \mathbb{R}^n, \quad x \in (-L,L), \tag{1}$$

implemented `pde2path`, with special attention to traveling waves. For a background on `pde2path` we refer to [UWR14, Uec17c], and for an introductory tutorial to [RU17]. In §3 we consider the class of Ginzburg-Landau equations

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

with real parameters $\ell, s, \gamma, r, \nu, c_3, \mu, c_5$ posed on the interval $x \in (-\pi, \pi)$ with periodic or homogeneous Neumann boundary conditions. For $\gamma = 0$ both boundary conditions imply a 'gauge' symmetry with respect to rotations $A \mapsto \mathrm{e}^{\mathrm{i}\varphi}A$, $\varphi \in [0, 2\pi)$, which is also present on $x \in \mathbb{R}$. Hence, for $\gamma = 0$ equation (2) is equivariant with respect to the action of the special orthogonal group $SO(2)$. Periodic boundary conditions imply a translation invariance as a typically additional $SO(2)$-equivariance. However, for wavetrains $A(t,x) = R\exp(\mathrm{i}(kx - \omega t))$ the rotation and translation have the same group orbits; here $R > 0$ is the amplitude, $\omega, k \in \mathbb{R}$ the frequency and wave number. Equation (2) may also have reflection symmetries: $x \to -x$ for $s = 0$, and $A \mapsto \overline{A}$ for $\nu = \mu = 0$.

In §3 we consider fronts in a FitzHugh-Nagumo (FHN) type system, and in addition to the continuation of equilibrium solutions, we discuss (numerical) time evolution with symmetry reduction, also known as 'freezing' [BT07, BORM14]. This is also used in §4 to compute traveling fronts for an autocatalysis model, and pulses for another FHN model, from [BCM99] and [IIM00], respectively. These fronts and pulses may undergo Hopf bifurcations, and thus we also explain a setup to compute the associated relative time periodic orbits with suitable average speeds.

The associated demo directories `cGL, fhn, modfro` and `breathe` are included in `symtut` under the `pde2path/demos` folder.

1.1 Phase conditions

A continuous symmetry of G in (1) with respect to a Lie group Γ implies that nontrivial solutions come in continuous families given by the group orbit $u(g)$, $g \in \Gamma$, which yields zero eigenvalue(s) of the linearisation of G in a solution. Hence, a parameter continuation approach for the steady state problem

$$0 = G(u),$$

which uses the Newton method requires to remove the symmetry in order to converge robustly. While in practice the discretization may break the symmetry, this is typically unreliable, leads to ill-conditioned matrices, and does not allow for a proper approximation of the continuous problem. A natural and practical selection of a (locally) unique element on the group orbit goes by adding a constraint that requires the predictor u from a solution $u_{\rm old}$ to lie transverse to the group orbit of $u_{\rm old}$. In a Hilbert space this is naturally an orthogonality relation, the so-called 'phase condition',

$$\langle \partial_g u_{\rm old}, u - u_{\rm old}\rangle = 0, \tag{3}$$

which corresponds to $\langle \partial_g u, \partial_\mu u\rangle = 0$ for the continuous problem with a parameter μ. Here $\partial_g u$ is determined from the group action and relates to the generators in the Lie-algebra. For a symmetry of translations in x-direction this gives $\partial_g = \partial_x$ and for the rotations of the Ginzburg-Landau equations $\partial_g = \mathrm{i}$, i.e., multiplication with i.

The generic implementation of phase conditions in `pde2path` works via auxiliary equations

$$Q_j(u,\lambda) = 0, \quad j = 1,\ldots,n_Q, \tag{4}$$

which are coupled to the 'PDE-part' $\partial_t u = -G(u)$, respectively $0 = G(u)$ for the steady case, see [RU17, §1] for more details. Adding an equation requires an additional parameter η in order to solve the combined problem with a Newton method, which may be viewed as the Lagrange multiplier associated to the constraint. The natural modification of G is to add the generator with new parameter η as

$$0 = G(u,\lambda) + \eta\partial_g u. \tag{5}$$

Viewed as a steady state problems with left hand side the time derivative $\partial_t u$, the parameter η thus measures the velocity with which the resulting solutions move through the group orbit. For traveling waves it is the speed in space, s in (2), and for rotating waves of (2) the gauge frequency, ν in (2). Note that a linear problem possesses a dilation symmetry by multiplying with complex numbers. Here the natural constraint is a fixed canonical scalar product $\langle u, u\rangle \in \mathbb{C}$ of the complex eigenfunction which yields the complex eigenvalue as the associated two-dimensional parameter.

A periodic domain and constant coefficients in G imply a translation symmetry that can be preserved on the discrete level as a discrete translation symmetry. However, for front solution the truncation to a bounded domain necessarily breaks the translation symmetry on $\mathbb{R}$ even though, e.g., homogeneous Neumann boundary conditions on a 'large' interval can be used to approximate the front. Imposing the phase condition then compensates for the lack of translation symmetry to determine the velocity of the front as a traveling wave. We remark that a better approximation for fronts as spatially heteroclinic orbits uses projection boundary conditions onto the spatial un/stable eigenspaces of the asymptotic states, i.e., Robin-type BC, but often the additional effort to implement these is not needed in practice.

1.2 Bifurcation with symmetry and pde2path

The detection of bifurcations is affected by symmetries, for instance in constant coefficient problems, periodic boundary conditions double the multiplicity of eigenvalues of the homogeneous Neumann boundary conditions through the translates of the Neumann eigenfunctions. In pde2path even multiplicity means that the usual branch point detection by checking sign changes in the determinant via LU-decomposition fails in case of periodic boundary conditions. In addition, the convergence of eigenvalues via the Matlab routine eigs can be problematic for multiple eigenvalues. The detection problem can often be solved by locating a change in the number of unstable eigenvalues via p.sw.bifcheck=2 (see [Uec17b, §2.1]), and warnings due to a poor convergence of eigs can sometimes be avoided by reducing p.nc.neigs – however, both of these ad hoc approaches should be used with caution.

Symmetries cause at least group orbits of branches to bifurcate, and branch switching at the resulting multiple zero eigenvalue, especially in 2D and 3D, requires judicious choice of the predictor to detect all bifurcating branches. One may also need to account for symmetries that are present in the linear problem only. While for continuous symmetries the phase conditions select a representative of the group orbit, phase conditions can generally be applied to non-trivial solutions only, and thus the initial step onto the branch requires caution as illustrated for (2) below.

Another symmetry that often occurs in applications is a Galilean boost invariance, which is usually broken by a mass constraint as a phase condition. See, e.g., [DRUW14, §2.4] for the so called functionalized Cahn-Hilliard equation, or [BGUY17] for a model for ionic liquids.

Here we restrict to 1D, but our codes naturally and easily extend to 2D or 3D, cf. [RU17, DU17] for simple stationary examples. The number of required phase conditions then depends on the type of BC (periodic BC in only one or in several directions) and the type of solutions (homogeneous or not in some directions).

2 Complex Ginzburg Landau equation: demo cGL

In terms of real and imaginary parts $A = u_1 + \mathrm{i}u_2$ equation (2) reads

$$\partial_t \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} \ell^2\partial_x^2+s\ell\partial_x+r & -\nu \\ \nu & \ell^2\partial_x^2+s\ell\partial_x+r \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} -|A|^2 \begin{pmatrix} c_3u_1-\mu u_2 \\ \mu u_1+c_3u_2 \end{pmatrix} -c_5|A|^4 \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} + \begin{pmatrix} \gamma \\ 0 \end{pmatrix}, \quad (6)$$

where $|A|^2 = (u_1^2+u_2^2)$. In this form the problem can be readily implemented in pde2path, cf. [Uec17b], where the problem with $\gamma = 0$ and $\nu \neq 0$ has been used as a test problem for Hopf bifurcations (under Neumann BC). For background on the implementation of periodic BC in pde2path we refer to [DU17], and here only recall that the basic procedure is to

- first set up the problem with Neumann BC;
- call box2per, which generates matrices drop and fill; these are then first used in an initial step to modify the mass and stiffness matrices, and secondly in the implementations of the right hand side G and its Jacobian to extend/reduce the solution across periodic boundaries.

Consequently, the implementation of G from (6) reads as in Listings 1, 2, while Listings 3, 4 give the (joint) implementations of the two phase-conditions

$$\mathrm{PC}_1\colon \langle \partial_x u_{\mathrm{old}}, u-u_{\mathrm{old}}\rangle = 0, \qquad \mathrm{PC}_2\colon \langle \mathrm{i}u_{\mathrm{old}}, u-u_{\mathrm{old}}\rangle = 0, \quad (7)$$

and their Jacobian(s). Table 1 gives an overview of the m-files in symtut/cGL.

```
function r=sG(p,u) % compute pde-part of residual
par=u(p.nu+1:end); f=nodalf(p,u);
r=par(9)^2*p.mat.K*u(1:p.nu)-p.mat.M0*f-par(9)*par(2)*p.mat.Kx*u(1:p.nu);
```

Listing 1: cGL/sG.m. The matrices M, M0 (which involves fill'), K and Kx are generated in oosetfemops, and the nonlinearity (everything without derivatives) is outsourced to nodalf, see Listing 2

Table 1: Scripts and functions in `symtut/cGL`.

script/function	purpose,remarks
cmds1,...,cmds3	scripts, periodic BC in cmds1, fold-continuation in cmds2, Neumann BC in cmds3, see also plotcmds.m.
cGLinit, oosetfemops	initialization function, and setting of FEM operators, including tranform to periodic domain.
sG, nodalf	rhs G, where the nonlinearity is computed in nodalf.
sGjac, njac	Jacobian $\partial_u G$, with the main computation in njac.
qf_sym, qfder_sym	the phase conditions (7), and the derivatives.
qf_trans, qfder_trans	only the first phase condition from (7), and its derivative.
spjac, spqjac	Jacobians $\partial_u(\partial_u G\phi)$ and $\partial_u(\partial_u q\phi)$ for fold continuation, see (8).
e2rs	gradient based elements 2 refine selector, used for mesh-refinement in cmds3.

```
function f=nodalf(p,u) % nonlinearity for cGL
par=u(p.nu+1:end); u=p.mat.fill*u(1:p.nu); % extract param., extend sol
r=par(1); nu=par(3); mu=par(4); c3=par(5); c5=par(6); gam=par(7);
n=p.np; u1=u(1:n); u2=u(n+1:2*n); ua=u1.^2+u2.^2; % aux variable |u|^2
f1=r*u1-nu*u2-ua.*(c3*u1-mu*u2)-c5*ua.^2.*u1 + gam;
f2=r*u2+nu*u1-ua.*(c3*u2+mu*u1)-c5*ua.^2.*u2;
f=[f1;f2];
```

Listing 2: `cGL/nodalf.m`. The nonlinearity for (6).

```
function q=qf_sym(p,u) % phase condition for both translation and rotation
par=u(p.nu+1:end); uox=p.mat.Kx*p.u(1:p.nu); q=uox'*(u(1:p.nu)-p.u(1:p.nu));
q=[q; (p.mat.R*p.u(1:p.nu))'*(u(1:p.nu)-p.u(1:p.nu))+par(8)];
```

Listing 3: `cGL/qf_sym.m`. The last solution $u_{\rm old}$ is stored in `p.u`, and the predictor is `u`. Line 2 computes `uox`=$\partial_x u_{\rm old}$ and `q`=$\langle\partial_x u_{\rm old}, u-u_{\rm old}\rangle_{L^2}$. Line 3 adds PC_2, where the rotation by i is implemented via `p.mat.R`, which is set in `oosetfemops.m`. The auxiliary `par(8)` is used in the cGL-demo for continuation in the group orbit of R.

```
function qu=qfder_sym(p,u) % phase condition jacs translation and rotation
qu=(p.mat.Kx*p.u(1:p.nu))'; qu=[qu; (p.mat.R*p.u(1:p.nu))'];
```

Listing 4: `cGL/qfder_sym.m`. The derivatives of the components of `q` defined in `cGL/qf_sym.m` from Listing 3 with respect to u.

We start by studying the case $s=\gamma=\nu=\mu=0$ with the aforementioned additional reflection symmetries; note that the combined reflection $A\to \mathrm{i}\overline{A}$ acts here as the permutation $(u_1,u_2)\mapsto(u_2,u_1)$. For these parameters, the linearization in the trivial steady state $u_1=u_2=0$ consists of two identical uncoupled equations so that any eigenvalue is geometrically at least double. On $\mathbb{R}$ or for periodic boundary conditions, the translation symmetry implies an additional symmetry of arbitrary relative translation between the components, which is however broken by the nonlinear terms.

2.1 Example results for periodic boundary conditions

First steps. For periodic boundary conditions, we find that the numerical predictor from `swibra`, after locating a bifurcation via `p.sw.bifcheck=2`, in a first step leads to a solution in the invariant subspace $u=v$. These are solutions of the Allen-Cahn equation for $A\in\mathbb{R}$ since $A=u\mathrm{e}^{\mathrm{i}\pi/4}$ with real u lies in the group orbit of $A=u\in\mathbb{R}$, which gives the Allen-Cahn equation. However, without a phase condition, further continuation steps jump in an uncontrolled way to the branch of wavetrain type $A=R\exp(\mathrm{i}kx)$, i.e., a relative translation by $\pi/4$: $u=\cos$, $v=\sin$. Space-shifting a solution on the Allen-Cahn branch can be used to do a controlled jump to wavetrains. The detailed commands used here and for further branch continuation are shown in Listing 5, and the results are plotted in Figures 1.

```
close all; format compact; keep pphome; % clean up
%% cell 1: init, and continuation of trivial branch
```

```
p=[]; lx=pi; nx=30; p=cGLinit(p,lx,nx); % initialize
% parameters: 1=r,2=speed,3=nu,4=mu,5=c3,6=c5,7=gam,8=symmetry,9=domain scale
par=[-1; 0; 0; 0; -1; 1; 0; 0; 1];
u=zeros(p.np,1); v=u; p.u=[u;v; par]; % initial guess (here trivial) and pars
dir='zero'; p=setfn(p,dir); p.fuha.savefu(p); % set dirname and save
%% cell 2: PERIODIC BC: continue zero solution on periodic domain
p=loadp('zero','pt0','per/zero'); p=box2per(p,1); % switch to periodic BC
p=cont(p,20); % continuation of (here) trivial branch, incl. bif-detec
%% cell 3 - switch to branch for a few steps without phase-cond (PC)
p=swibra('per/zero','bpt2','per/ini',0.025); %p.usrlam=[];
p.sw.bifcheck=0; p.sw.spcalc=0; % switch off bif-detection (is random without PC)
p.file.smod=1; p=cont(p,1); % do 1 initial step (for check), and store all
%% cell 3a - controlled step to wavetrain branch
p.u(1:p.nu/2)=circshift(p.u(1:p.nu/2),round(p.nu/8));
p.u=nloop(p,p.u); plotsol(p); p.fuha.savefu(p);
%% cell 4 - add phase conditions to 1st point of cell 3 and continue
p=loadp('per/ini','pt1','per/stand'); p=resetc(p); p.file.smod=10;
p.nc.ilam=[1;2;3];  p.nc.nq=2; % 2 phase-cond, speed and nu as add parameters
p.fuha.qf=@qf_sym;      % function handle for aux. eqn.
p.sw.qjac=1; p.fuha.qfder=@qfder_sym; % analytical jac for aux. eqn.
p.sw.bprint=2; clf(2); p.nc.dsmax=0.05;  p=cont(p,30);
%% cell 5 - add only transl. PC to 1st point of cell 3 -> uncontr. rotations
p=loadp('per/ini','pt1','per/rot0'); p=resetc(p); p.file.smod=10;
p.nc.nq=1; p.nc.ilam=[1;2];  % 1 phase-cond, speed as second parameter
p.fuha.qf=@qf_trans; p.sw.qjac=1; p.fuha.qfder=@qfder_trans;
p.sw.bprint=2; clf(2); p.nc.dsmax=0.05; p.sol.ds=-0.05; p=cont(p,40);
%% cell 6 - point of cell 3a (TW), only transl phase-cond sufficient
p=loadp('per/ini','pt2','per/wtstand'); p=resetc(p); p.file.smod=10;
p.nc.nq=1; p.nc.ilam=[1;2];  % 1 phase-cond, speed as second parameter
p.branch=[bradat(p); p.fuha.outfu(p,p.u)]; figure(2); clf;
p.fuha.qf=@qf_trans; p.sw.qjac=1; p.fuha.qfder=@qfder_trans;
p.sw.bprint=2; p.nc.dsmax=0.1; p.sol.ds=-0.1; p=cont(p,20);
%% Plotting cmds in plotcmds.m
```

Listing 5: `cGL/cmds1.m`. Cell 1: Initialization at trivial solution; in `cGLinit` we set `p.sw.bifcheck=2`. See Cell headings for purposes of further cells, and Figure 1 for some results (plots in `plotcmds.m`). The phase condition `qf_trans` in Cells 5,6 is `q=(p.mat.Kx*p.u(1:p.nu))'*(u(1:p.nu)-p.u(1:p.nu))+par(8);` i.e., only PC_1.

Fold continuation with constraints. Fold continuation without constraints ($n_q = 0$) is described in various settings in [RU17]. For $n_q \geq 1$ the extended system for $U = (u, \phi, w)$, where ϕ is in the kernel of $\partial_u G$ and $w = (\lambda, \eta)$ is a shorthand for the active parameters, reads

$$H(U) = \begin{pmatrix} G(u,w) \\ \partial_u G(u,w)\phi \\ q(u,w) \\ \partial_u q(u,w)\phi \\ \|\phi\|_{L^2}^2 - 1 \\ p(U) \end{pmatrix} = 0, \quad \text{with Jacobian } D_U H(U) = \begin{pmatrix} \partial_u G & 0 & \partial_w G \\ \partial_u(\partial_u G\phi) & \partial_u G & \partial_w(\partial_u G\phi) \\ \partial_u q & 0 & \partial_w q \\ \partial_u(\partial_u q\ \phi) & \partial_u q & \partial_w(\partial_u q\ \phi) \\ 0 & 2\phi^T & 0 \\ \partial_u p & \partial_\phi p & \partial_w p \end{pmatrix}. \tag{8}$$

Note that the last column of $D_U H(U)$ has dimension $2(n_u + n_q + 1) \times (2 + 2n_q)$. The last line of $D_U H(U)$ is generated automatically, and also $\partial_w G$ and $\partial_w(\partial_u G\phi)$ are obtained quickly by finite differences. For 1D model problems, this also works for $\partial_u(\partial_u G\phi)$ and $\partial_u(\partial_u q\phi)$, but in particular in higher space dimension for efficiency it is highly recommend to implement functions returning these objects. For a 2-component semilinear system such as (6), we have

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

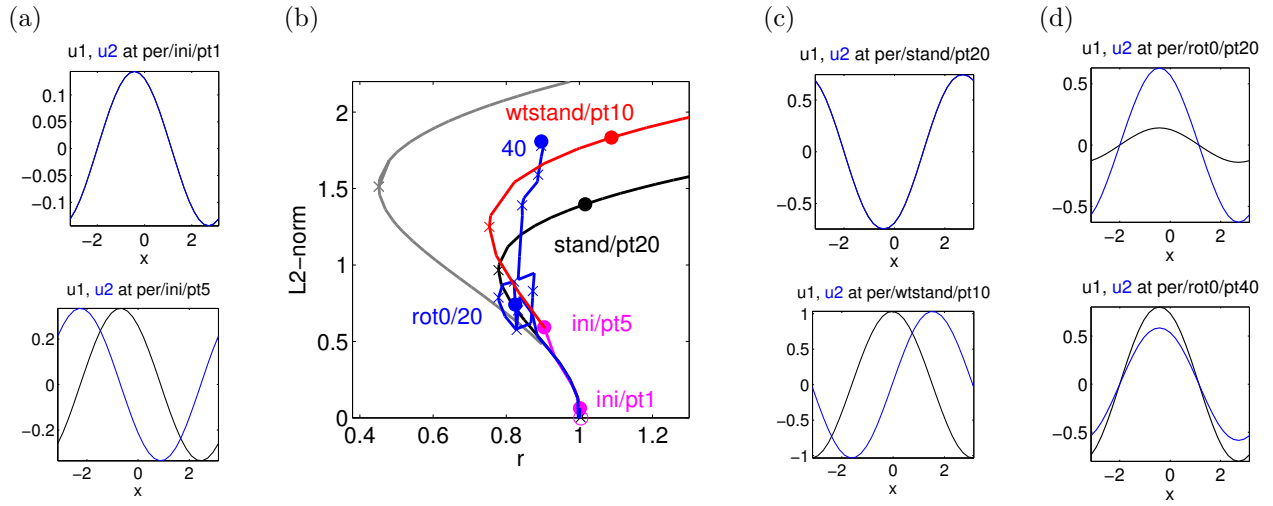

Figure 1: Plots of results for the cGL equation with periodic BC from `cmds1.m` (Listing 5). (a) Solution plots from the initial continuation of the 2nd branch without phase conditions. Here both components are plotted due to the setting `p.plot.pcmp=[1 2]; p.plot.cl={'black','blue'};` in `cGLinit.m`. The first step lies on the subspace of Allen-Cahn solutions with $u = v$, while at some step we jump to the subspace of wavetrains with $A = R\exp(ikx)$. (b-d): The black branch in (b) with $u = v$, with example solution in the top panel of (c), is obtained from switching on *both* phase conditions at `ini/pt1`. If at this point we only switch on the translational PC_1, then we obtain the blue branch, with uncontrolled rotations of the phase (example plots in (d)) – this is meant to illustrate problems that typically arise in continuation *without* proper phase conditions. On the other hand, only switching on PC_1 at the wave train solution `ini/pt5` yields the red branch of standing wave trains, because in this case PC_1 and PC_2 have the same group orbits. The gray line is obtained via fold continuation in c_5 of the black fold near $r = 0.8$ (cmds2.m).

and this is straightforwardly implemented in `spjac.m`, see the source code. Since the constraints in Listing 4 are linear in u, we moreover have $\partial_u(\partial_u q\ \phi) = 0 \in \mathbb{R}^{2\times n_u}$, see Listing 6. Cell 2 of Listing 7 then gives an example of fold continuation; Cell 3 shows how to return to regular continuation from a fold-point, and the Gray branch in Fig. 1(b) illustrates the results.

```
function quuph=spqjac(p,u) % \pa_u(q_u*phi), needed for fold continuation
quuph=sparse(p.nc.nq,p.nu);% here just 0-matrix
```

Listing 6: `cGL/spqjac.m`. $\partial_u(\partial_u q\phi)$ for q from Listing 4.

Continuation along group orbits. In Cells 3-6 of Listing 7 we continue along group orbits and illustrate some symmetry breaking, see the Listing caption for comments, and Fig. 2 for results.

```
%% cell 1 - fold-continuation
p=spcontini('per/stand','fpt2',6,'per/fc1'); % init fold cont, par 6 new prim. par
p.plot.bpcmp=1; figure(2); clf; p.sol.ds=-0.01; % use this new param.for plotting
p.sw.spjac=1; p.fuha.spjac=@spjac; % spectral jac
p.sw.spqjac=1; p.fuha.spqjac=@spqjac; % and jac for constraints
tic; p=cont(p); toc
%% cell 2 - switch back to regular cont
p=spcontexit('per/fc1','pt20','per/stand2a'); p.plot.bpcmp=0;
clf(2); p=cont(p,1); p=cont(p,20); % cont in one direction, 1 init.step for saving
% continue in other direction
p=loadp('per/stand2a','pt1','per/stand2b'); p.sol.ds=-p.sol.ds; p=cont(p,20);
%% cell 3 - continue standing wave in family given by rotation symmetry
p=loadp('per/stand','pt20','per/rot'); p=resetc(p); p.file.smod=1;
p.nc.ilam=[8;2;3]; p.nc.dsmax=0.3; p.nc.lammax=10; p=cont(p,30);
%% cell 4 - continue WT in family given by rot=transl symmetry (shape doesn't
    change)
p=loadp('per/wtstand','pt20','per/trans'); p=resetc(p); clf(2);
p.nc.ilam=[8;2]; p.nc.lammax=10; p=cont(p,20);
%% cell 5 - continue WT to nonzero speed/rotation (shape doesn't change)
```

```
p=loadp('per/wtstand','pt20','per/wtmove'); p=resetc(p); clf(2);
p.nc.ilam=[4;2]; p.sol.ds=-p.sol.ds; p.plot.bpcmp=2; p=cont(p,20);
%% cell 6 - break rotation symmetry through gamma for a wavetrain
p=loadp('per/wtstand','pt20','per/wtasym'); p=resetc(p); clf(2);
p.sw.foldcheck=0;  p.nc.ilam=[7;2]; p.plot.bpcmp=0; p=cont(p,20);
```

Listing 7: `cGL/cmds2.m`. Cells 1 and 2 perform fold continuation. Cell 3 and Fig. 2 (a,b) illustrate the effect of continuation along the group orbits of rotations. Cell 4 illustrates that for wavetrains continuation in `par(8)` is equivalent to continuation in position, while for instance continuation in μ adapts the speed; in both cases, the shapes of the wavetrains stay fixed. Finally, Cell 6 breaks the rotation symmetry by changing γ and yields asymmetric solutions.

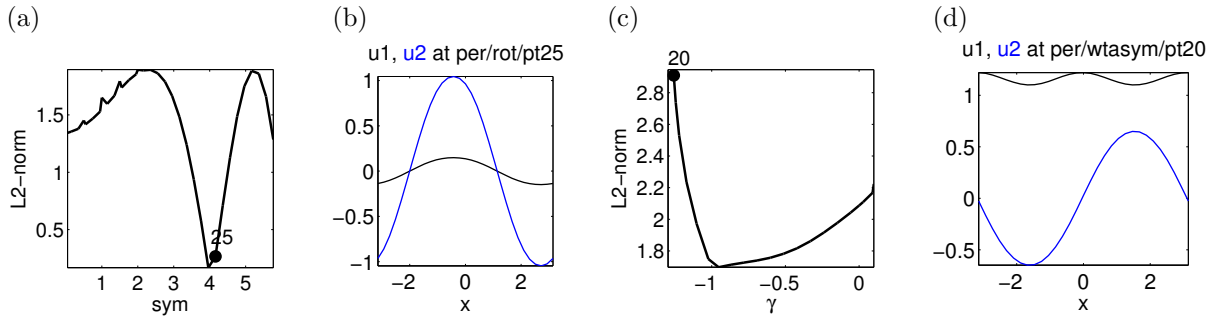

Figure 2: Some results for the cGL with periodic BC from Cells 3 and 6 of Listing 7.

2.2 Neumann boundary conditions

Neumann BC break the translation symmetry and the additional translation symmetry of the linearization in the trivial zero solution. However, bifurcation points from the zero solution still come with double zero eigenvalue due to the decoupled nature of the linearized equations. Hence, branch points should be detected via `p.sw.bifcheck=2`.

We focus on the branch that relates to fronts. As for periodic BC, branch switching initially goes without imposing a phase condition. The numerical predictor again steps onto the Allen-Cahn branch with $u = v$ and stays on it since in the present case there is no wavetrain branch. Listing 8 shows the commands we use in this case. As mentioned in the introduction, even though the translation symmetry is broken, proper approximation of traveling fronts from the unbounded domain requires adding the translation phase condition.

```
%% cell 1 - hom. NEUMANN BC: continue zero solution
p=loadp('zero','pt0','nbc/zero'); p=setlam(p,-0.1); p.nc.dsmax=0.1; p=cont(p,10);
%% cell 2 - switch to front-mode for one step
p=swibra('nbc/zero','bpt2','nbc/ini',0.01); p.sw.bifcheck=0; p=cont(p,1);
%% cell 3 - add phase conditions and continue
p=loadp('nbc/ini','pt1','nbc/stand'); p=resetc(p);
p.nc.ilam=[1;2;3]; p.nc.nq=2; p.fuha.qf=@qf_sym;
p.sw.qjac=1; p.fuha.qfder=@qfder_sym;
p.sw.bprint=[2;3]; clf(2); p.nc.dsmax=0.5; p=cont(p,20);
%% cell 4 - continue in family given by rotation symmetry
p=loadp('nbc/stand','pt20','nbc/rot'); p=resetc(p); clf(2);
p.nc.ilam=[8;2;3]; p=cont(p,20);  % L2-norm is of the first component only
%% cell 5 - increase domain size and refine mesh
p=loadp('nbc/stand','pt20','nbc/dom'); p=resetc(p); clf(2);
p.nc.ilam=[9;2;3]; p.plot.bpcmp=9;
p.sol.ds=-0.1; p.nc.dsmax=0.1; p.nc.lammin=0.5; p=cont(p,10);
p=meshada(p,'ngen',3,'sig',1e-4);
p.nc.dsmax=0.1; p.nc.lammin=0.25; p=cont(p,10);
%% cell 6 - nonzero frequency/speed through gamma
p=loadp('nbc/dom','pt10','nbc/move-'); p=resetc(p); clf(2);
p.nc.ilam=[7;2;3]; p.branch=[bradat(p); p.fuha.outfu(p,p.u)];
p.nc.lammin=-1; p.plot.bpcmp=2; p=cont(p,22);
```

```
p=loadp('nbc/dom','pt10','nbc/move+'); p=resetc(p); p.sol.ds=-p.sol.ds;
p.nc.ilam=[7;2;3]; p.branch=[bradat(p); p.fuha.outfu(p,p.u)];
p.nc.lammin=-1; p.plot.bpcmp=2; p=cont(p,22);
```

Listing 8: `cGL/cmds3.m` Cell 1 detects the bifurcation points, cell 2 goes one step without phase condition on the front-mode. Cell 3 adds both phase conditions for rotation and translation and computes the branch. Cell 4 continues in the group orbit of rotations, and Cell 5 continues in the domain size, refines the mesh, and continues further. Cell 6 illustrates continuation in the parameter γ; see Fig. 3 for results.

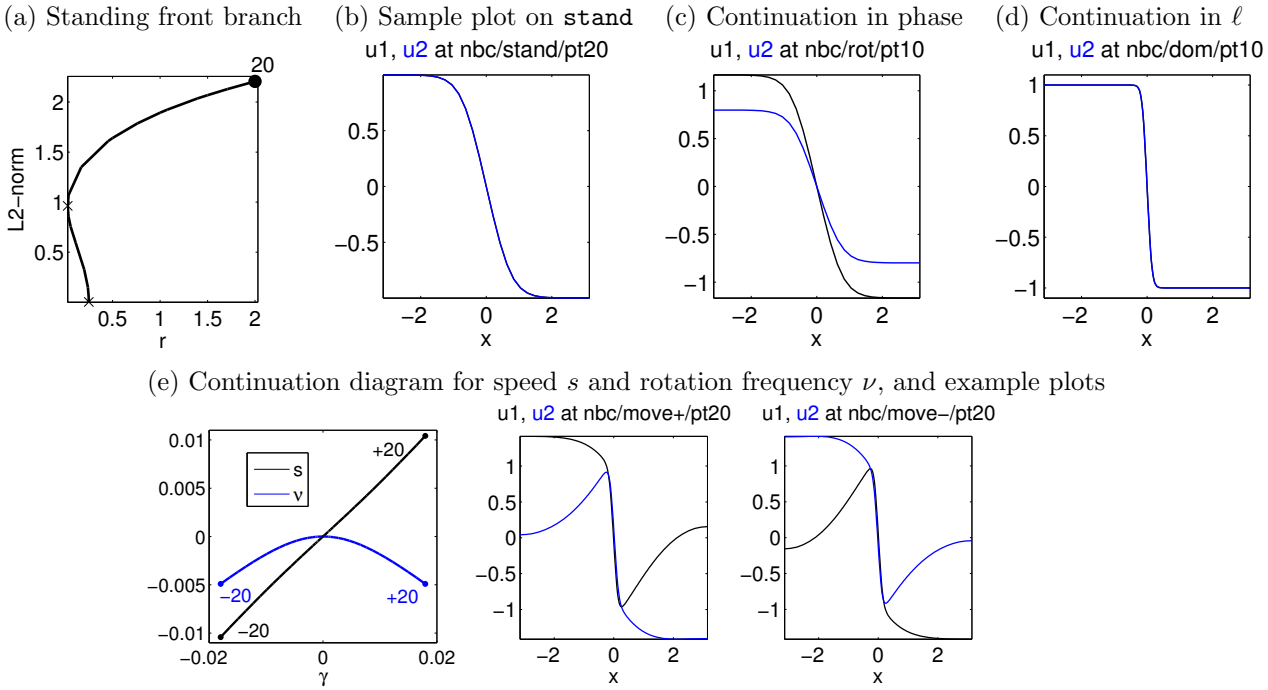

Figure 3: Plots of results for the cGL equation with Neumann BC from Listing 8. Cell 3 (a,b), Cell 4 (c), and Cell 5 (d), and Cell 6 (e).

Remark 2.1 In the above experiments, after the primary bifurcation from $(u, v) \equiv (0, 0)$ bifurcation detection was turned off since the secondary bifurcations in (6) are of Hopf type, which we do not consider here. Hopf bifurcations with phase constraints are discussed in §4 and [Uec17a]. ⌋

3 Fronts in a FitzHugh-Nagumo type model: demo `fhn`

In this section we consider front solutions in the FitzHugh-Nagumo type system in 1D

$$\begin{aligned} u_t &= \varepsilon^2 u_{xx} + s u_x + u - u^3 - \varepsilon(p_3 + p_4 v + p_5 v^2 + p_6 v^3) \\ v_t &= \varepsilon^2 (v_{xx} + u - v) + s v_x, \end{aligned} \tag{10}$$

$x \in \mathbb{R}$, which is implemented in `sG.m, nodalf.m` and the Jacobian in `sGjac.m` of the demo directory `fhn`. In addition to the translation symmetry in x and reflection symmetry in x for $s = 0$, the system possesses the reflection symmetry $(u, v) \mapsto -(u, v)$ for $p_3 = p_5 = 0$.

On the spatial scale x/ε, at $\varepsilon = 0$ (10) reduces to the symmetric Allen-Cahn equation for u (and $v_{xx} = 0$), which possesses stationary front solutions. As shown in [CBvHR12], for $0 < \varepsilon \ll 1$ these steady fronts perturb to slowly moving fronts with velocity s of order ε^2 and sharp interface, whose existence is to leading order in ε equivalent to

$$p_3 + p_4 v(s) + p_5 v(s)^2 + p_6 v(s)^3 - \frac{\sqrt{2}}{3} s = 0,$$

where $v(s) = \frac{s}{\sqrt{s^2+4}}$. This equation has a cusp singularity at $p_4 = 2\sqrt{2}/3$ and $p_3 = p_5 = p_6 = 0$ and the associated center manifold captures the dynamics of the front velocity as discussed in [CBDvHR15].

An overview of the scripts and functions in `symtut/FHN` is given in Table 2. The commands in `cmds1` (Listing 9) compute the standing and traveling fronts with `pde2path` by continuation from the zero state at $s = p_j = 0$, $j = 3, \ldots, 6$ with homogeneous Neumann boundary conditions and phase condition of translation symmetry as in the previous section; here implemented in `qf.m` with Jacobian `qfder.m`.

Table 2: Scripts and functions in `symtut/FHN`.

script/function	purpose,remarks
cmds1	script for computing the steady and traveling fronts by bifurcation
cmds2	convergence to traveling front by freezing simulation starting from perturbations of unstable steady fronts
cmds3	fold continuation to obtain cusp in p_4
fhnplot	script with plotting commands
FHNinit, oosetfemops	initialization function, and set up of FEM operators
sG, nodalf, sGjac	rhs G, where the nonlinearity is computed in nodalf, and Jacobian
qf, qfder	the translational phase condition $\langle \partial_x u_{\rm old}, u\rangle = 0$, and its u-derivative
spjac, spqjac	Jacobians $\partial_u(\partial_u G\phi)$ and $\partial_u(\partial_u q\phi)$ for fold continuation.
e2rs	gradient based elements 2 refine selector, used for mesh-refinement in cmds3
tintfreeze	function for time integration with "freezing".

```
%% demo fhneps, clear,  - init and findbif
close all; format compact; keep pphome; p=[]; p=FHNinit(p,10,50);
%pars: 1=eps; 2=vel; others of coupling function g=p_3+p_4*u+p_5*u^2+p_6*u^3
par=[6; 0; 0; 0; 0; 0;]; u=zeros(p.np,1); v=u; p.u=[u;v; par]; p=findbif(p,1);
%% cell 1 - swibra to stationary front for one step
p=swibra('init','bpt1','prep/swibra',0.1); p.nc.lammin=0.5; p=cont(p,1);
%% cell 2 - add velocity and phase equation
p=swiparf('prep/swibra','pt1','prep/decoup',[1;2]);
p.nc.nq=1; p.nc.xiq=0.1; p.fuha.qf=@qf; p.fuha.qfder=@qfder;
p.sol.ds=-0.1; p.nc.lammin=0.1; p.usrlam=[0.4,0.3,0.2]; p.sw.bifcheck=0;
p.nc.dsmax=0.5; p.nc.dlammax=0.5; p.sw.bprint=2; clf(2); p=cont(p,30);
%% cell 3 - refine mesh
p=loadp('prep/decoup','pt20','prep/decoup');
p=meshada(p,'ngen',3,'sig',1e-4);  p.fuha.savefu(p);
%% cell 4 - turn on cubic coefficient
p=swiparf('prep/decoup','pt21','prep/cubic',[6;2]);
p.nc.lammin=-1; p.nc.lammax=1;
p.usrlam=[0.25,0.5,0.75]; p.sol.ds=-0.1; clf(2); p=cont(p,20);
%% cell 5 - find bifurcation point
p=swiparf('prep/cubic','pt5','stand',[4;2]); p.sw.bifcheck=1;
p.nc.lammin=-2; p.nc.lammax=1.5; p.sol.ds=0.1;
p.usrlam=[]; clf(2); p=cont(p,25); plotsol(p,6);
%% cell 6 - switch to branch with nonzero velocity
p=swibra('stand','bpt1','travel',0.1); p.nc.dsmax=0.1;
p.plot.bpcmp=2; clf(2); p=cont(p,20);
%%
p=swibra('stand','bpt1','travel-',-0.01); p.nc.dsmax=0.1;
p.plot.bpcmp=2; clf(2); p=cont(p,20);
```

Listing 9: `fhn/cmds1.m`. Commands to locate the front solutions of (10) and compute a supercritical pitchfork bifurcation in the front velocity. In Cell 1 we find a primary bifurcation from $(u, v) \equiv 0$ at relatively large ε with coupling parameters equal to 0. In Cell 2 we continue the decoupled front branch down to $\varepsilon = 0.1$, where we obtain a rather sharp interface, calling for the mesh-refinement in Cell 3. In Cell 4 we continue in the cubic coefficient $p_6 =$ `par(6)` and then in Cell 5 find a bifurcation point upon continuation in $p_3 =$ `par(4)`, leading to moving fronts in Cell 6. See Figure 4.

Due to the spatial scale separation by ε, the fronts are sharp for small ε so that the mesh needs

to be adapted. Here we use an error estimator based on the gradient of the first component as shown in Listing 9.

```
function [p,idx]=e2rs(p,u) % elements to refineme selector
% here not via error estimation but simply via gradient
ux=p.mat.M\(p.mat.Kx*u(1:p.nu)); idx=find(abs(ux(1:p.np-1))>=p.nc.sig);
if(mod(p.np,2)); idx=idx(1:length(idx)-1); end;
```

Listing 10: `fhn/e2rs.m`. A mesh refinement strategy that respects the reflection symmetry.

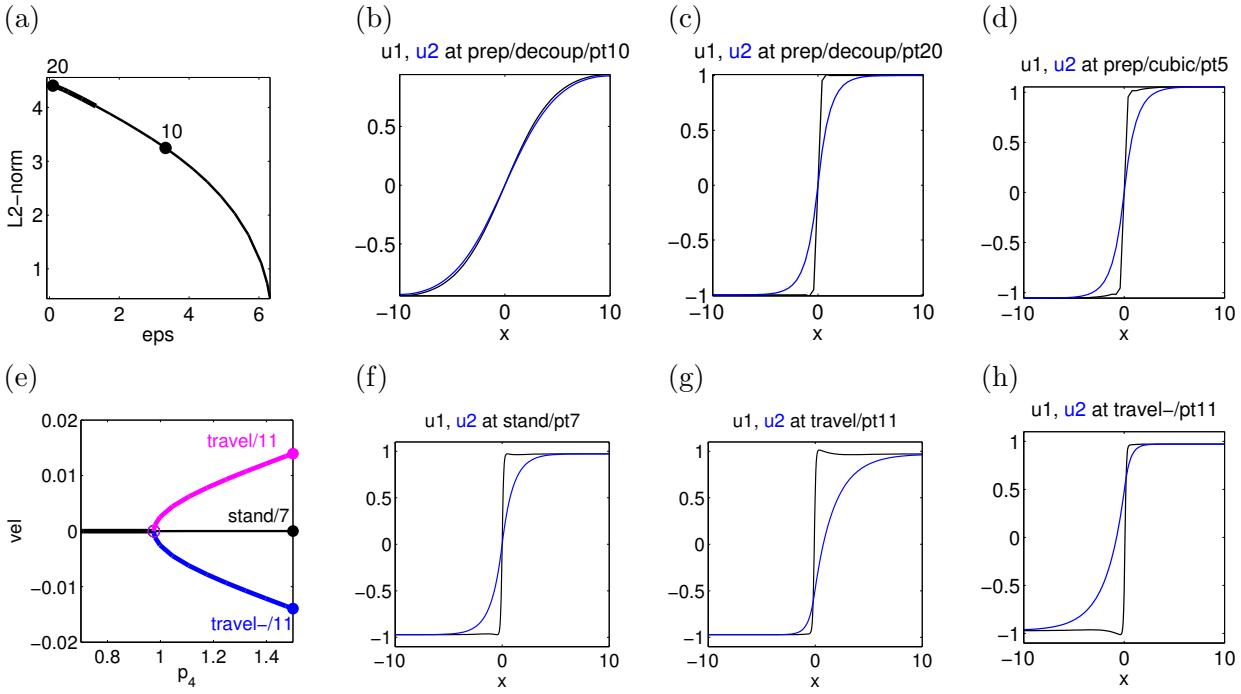

Figure 4: Plots of results for the FHN system from Listing 9. Decoupled branch (a), solution plots at $\varepsilon \approx 3.3$ (b) and $\varepsilon = 0.1$ (c), and solution after continuing (c) to $p_6 = -1$ (d). (e): supercritical pitchfork of fronts in terms of velocity and parameter p_3 (e), and profiles of stationary symmetric and asymmetric moving front at labeled points in (f-h).

Next, the dynamics on the aforementioned center manifold is computed with `pde2path` via a 'frozen' simulation of the parabolic equation, which removes the translational motion using the phase condition – now applied to the time stepping algorithm so that in (3) u_{old} is the previous time step. The location of the front-type solution is thereby effectively fixed in the mesh, which also keeps the sharp gradient at the refined part. Let $F(u, v)$ denote the right hand side of (10) for $s = 0$. Then at each time step t the current 'speed' $s = s(t)$ relative to the translation mode (u_x, v_x) is determined as the projection of $-F$ onto (u_x, v_x):

$$s(t) = -\frac{\langle F(u,v), (u_x, v_x)\rangle}{\|(u_x, v_x)\|_2^2},$$

which is equivalent to the phase condition $\langle (u_t, v_t), (u_x, v_x)\rangle = \langle F(u,v) + s(t)(u_x, v_x), (u_x, v_x)\rangle = 0$, i.e., the time evolution it orthogonal to the translation symmetry, cf. (3).

The implementation of this adapted velocity requires just a small change of a time integration loop and the routine `tintfreeze` is shown in Listing 11, based on the routine `tints`. The vector `vel` is returned by this routine in order to be processed for plotting or further evaluation, see Listing 12 and Figure 5.

```
% adapted from tint to freeze the translation symmetry
% additional input/output vel=[times(..); speeds(...)]
```

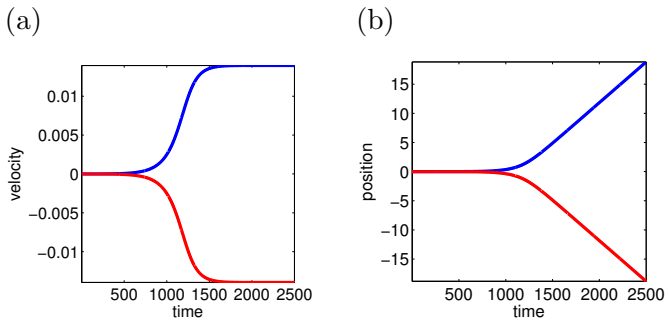

Figure 5: Plots of (a) the velocity `vel(n)` and, by integration, positions (b) for the simulation with `tintfreeze` of two perturbations from an unstable front from Listing 11.

```
% additional input vmod=skip for augmenting vel
function [p,t1,vel]=tintfreeze(p,t1,dt,nt,pmod,vel,vmod)
n=0; t=t1; K=p.u(p.nu+1)^2*p.mat.K; Kx=p.mat.Kx;
% prefactor stiffness matrix for semi-implicit time-stepping:
Lam=p.mat.M+dt*K; [L,U,P,Q,R]=lu(Lam);
while(n<nt) % integration loop
  f=nodalf(p,p.u);  uKx = Kx*p.u(1:p.nu);
  r=K*p.u(1:p.nu)-p.mat.M*f; cs=(uKx'*r)/(uKx'*uKx); % cs=current speed
  g=p.mat.M*p.u(1:p.nu)+dt*(p.mat.M*f+cs*uKx);
  p.u(1:p.nu)=Q*(U\(L\(P*(R\g)))); n=n+1; t=t+dt; % time stepping:
  if (mod(n,vmod)==0);  vel=[vel [t; cs]]; end  % for returning velocity
  if(mod(n,pmod)==0); plotsol(p,p.plot.ifig,p.plot.pcmp,p.plot.pstyle); end
end
t1=t;
```

Listing 11: `fhn/tintfreeze.m`. Sample of time (linearly implicit) stepping that freezes the translation symmetry. Note that `tintfreeze` can be called repeatedly, if an initial call turned out to cover a too short time interval, returning a growing vector `vel` of times and speeds. Also note that in l5 we explicitly define the effective stiffness matrix, which we then prefactor in l7, while the nonlinearity is computed in l9 via `nodalf`. Thus, `tintfreeze` is somewhat problem dependent (see also the implementations in the `modfro` and `breather` directories).

```
%% cell 1 - perturb by eigenfunction and simulate time dynamics
p=loadp('stand','pt7','sim'); p.nc.nq=0; [muv,V]=specGu(p); muv(1:4)
p.u(1:p.nu)=p.u(1:p.nu)-1e-3*real(V(1:p.nu,2));
vel1=[]; t1=0; nt=5000; dt=0.5; pmod=500; vmod=50;
[q,t1,vel1]=tintfreeze(p,t1,dt,nt,pmod,vel1,vmod);
t1=0; vel2=[]; p.u(1:p.nu)=p.u(1:p.nu)+2e-3*real(V(1:p.nu,2)); % other direction:
[q,t1,vel2]=tintfreeze(p,t1,dt,nt,pmod,vel2,vmod);
%% cell 2 - velocity plot
figure(7); clf; fs='fontsize';
plot(vel1(1,:),vel1(2,:),vel2(1,:),vel2(2,:),'-r','LineWidth',3);
xlabel('time',fs,16); ylabel('velocity',fs,16); set(gca,fs,16); axis tight
%% cell 3 - position plot
tl=size(vel1,2); pos1=zeros(1,tl); pos2=pos1;
for(i=1:tl-1)
    pos1(i+1)=pos1(i)+vel1(2,i)*(vel1(1,i+1)-vel1(1,i));
    pos2(i+1)=pos2(i)+vel2(2,i)*(vel2(1,i+1)-vel2(1,i));
end;
figure(8); clf; plot(vel1(1,:),pos1,vel2(1,:),pos2,'-r','LineWidth',3)
xlabel('time',fs,16); ylabel('position',fs,16); set(gca,fs ,16);axis tight
```

Listing 12: `fhn/cmds2.m` Simulation from unstable front in supercritical pitchfork plotted in Figure 4. Results are shown in Figure 5.

Next we break the reflection symmetry $(u, v) \mapsto -(u, v)$ by changing the parameter p_3, which relates to unfolding the cusp singularity at the pitchfork bifurcation point. The commands in combination with time simulation from an asymmetric unstable front are shown in Listing 13 and the results are plotted in Fig. 6.

```
%% cell 1 - use p3 to break symmetry to obtain direction reversing front
p=swiparf('stand','pt7','asym',[3;2]); p.nc.lammin=-0.4; p.nc.lammax=1; clf(2);
p.usrlam=0.05; p.sol.ds=0.1; p.sw.foldcheck=1; p.plot.bpcmp=2; p=cont(p,10);
%% cell 2 - continue the fold to get the underlying cusp
p=spcontini('asym','fpt1',4,'cusp'); p.plot.bpcmp=p.nc.ilam(2); clf(2);
p.sw.spjac=1; p.fuha.spjac=@spjac; p.sw.spqjac=1; p.fuha.spqjac=@spqjac;
p.sol.ds=-0.01; p.nc.lammax=1.5; p=cont(p);
%% cell 3 - perturb by eigenfunction and time-integrate
p=loadp('asym','pt2','sim'); [muv,V]=specGu(p); n=p.nu;
p.u(1:n)=p.u(1:n)+1e-3*real(V(1:n,1)); vel1=[]; t1=0; nt=5000; dt=0.5; pmod=500;
    vmod=50;
[q,t1,vel1]=tintfreeze(p,t1,dt,nt,pmod,vel1,vmod);
p.u(1:n)=p.u(1:n)-2e-3*real(V(1:n,1)); vel2=[]; t1=0; % other direction
[p,t1,vel2]=tintfreeze(p,t1,dt,nt,pmod,vel2,vmod);
p=setaux(p,2,vel2(2,end)); p.fuha.savefu(p);
%% cell 4 - velocity and position plots
figure(7); clf; fs='fontsize';
plot(vel1(1,2:end),vel1(2,2:end),vel2(1,2:end),vel2(2,2:end),'-r','LineWidth',3);
xlabel('time',fs,16); ylabel('velocity',fs,16); set(gca,fs,16); axis tight
tl=size(vel1,2); pos1=zeros(1,tl); pos2=pos1;
for(i=1:tl-1)
    pos1(i+1)=pos1(i)+vel1(2,i)*(vel1(1,i+1)-vel1(1,i));
    pos2(i+1)=pos2(i)+vel2(2,i)*(vel2(1,i+1)-vel2(1,i));
end;
figure(8); clf; plot(vel1(1,:),pos1,vel2(1,:),pos2,'-r','LineWidth',3)
xlabel('time',fs,16); ylabel('position',fs,16); set(gca,fs,16); axis tight
%% cell 5 - continue final state as TW (get branch from cell 1 again!)
p=swiparf('sim','pt3','travel2',[3;2]); p.usrlam=p.u(p.nu+3); p.nc.lammin=-4;
p.nc.lammax=0.2; p.sol.ds=0.1; p.nc.dsmax=0.1; clf(2); p=cont(p,30);
figure(3); clf; plotbra(p,3,2,'lsw',0);
```

Listing 13: `fhn/cmds3.m`. Commands for continuation of Fig. 4(f) in $p_3 =$ `par(3)` (Cell 1), continuation of the obtained fold (Cell 2), yielding the unfolding of the cusp, for simulation of perturbations from an asymmetric unstable front (Cells 3 and 4). If after time integration we again continue in $p_3 =$ `par(3)` (Cell 5), then we regain the branch from Cell 1.

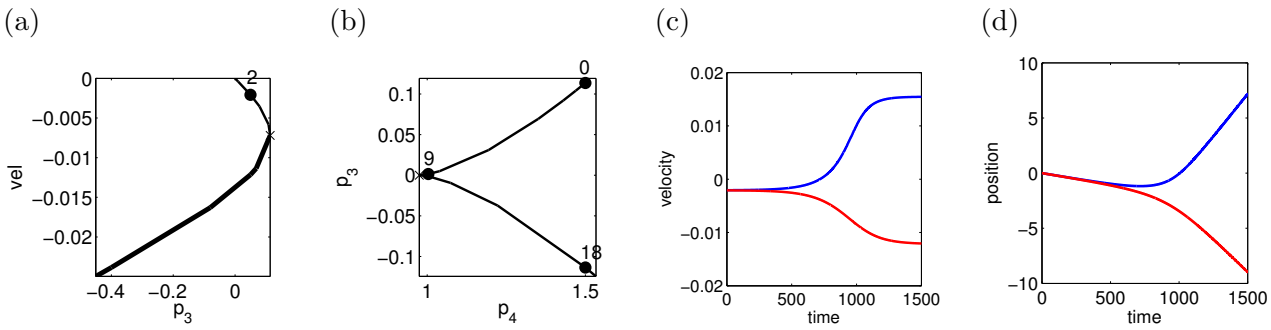

Figure 6: Results from Listing 13. (a) continuation in p_3. (b) continuation of the first fold on asymmetric branch through the cusp point. (c,d) plots of the velocities `vel(n)` and, by integration, positions for simulations with `tintfreeze` of two perturbations from the asymmetric unstable front with label 2 in (a).

4 Hopf–bifurcation with symmetry

As examples of Hopf bifurcations from traveling (and standing) waves we consider two models from [BCM99] and [IIM00]. The first, also considered in [BT07, Examples 3.1 and 5.5], reads

$$\partial_t u = a\partial_x^2 u - uf(v), \quad \partial_t v = \partial_x^2 v + uf(v), \tag{11}$$

$f(v) = v^m$ for $v \geq 0$ and 0 otherwise, where $a > 0$ and $m \geq 2$. are parameters. We study (11) on the domain $\Omega = (-l_x, l_x)$ with sufficiently large l_x, and with BC $(u, v) = (u, v)_{\mp}$ for $x = \mp l_x$, with $(u, v)_- = (0, 1)$ and $(u, v)_+ = (1, 0)$ which fixes a boost invariance of (11). Using freezing, we first find a TW for a near 0.2 with (constant) speed s, and then lowering a we find a Hopf bifurcation to a modulated TW.

The second example is similar to the FHN equation from §3, namely

$$\partial_t u = \partial_x^2 u + f(u, v), \quad \partial_t v = D\partial_x^2 v + g(u, v), \tag{12}$$

with homogeneous Neumann BC, $f(u, v) = u(u - \alpha)(\beta - u) - v$, $g(u, v) = \delta(u - \gamma v)$, with $\alpha, \beta, \gamma > 0$, and $0 < \delta \ll 1$. This model has standing and traveling pulses (and traveling fronts), and for $\delta \to 0$ we find a Hopf bifurcation to standing (and traveling) breathers. See also [HM94, GF13] for related models and results.

The basic strategy for both models is the same, and quite related to §3: first use freezing to converge to an appropriate traveling wave, then add the phase condition $0 = q(u) := \langle \partial_x u_0, u - u_0 \rangle$. In contrast to the examples from §2 and §3 we now 'freeze' by using a fixed reference profile u_0, and consider the extended system

$$M\partial_t u = -(G(u) - s\partial_x u), \tag{13a}$$
$$0 = q(u). \tag{13b}$$

We continue stationary solutions (corresponding to TW) of (13) in some parameter, including adaptive mesh refinement to deal with sharp interfaces, and find a Hopf bifurcation point (HBP) for the extended system. For this we naturally use the (generalized) eigenvalue problem

$$\begin{pmatrix} \mu M & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v \\ \sigma \end{pmatrix} = \begin{pmatrix} -(\partial_u G(u) - s\partial_x) & \partial_x u \\ \partial_u q(u) & 0 \end{pmatrix} \begin{pmatrix} v \\ \sigma \end{pmatrix}, \tag{14}$$

where a zero eigenvalue from (approximate) translational invariance of (11) and (12) is absent.

However, the additional algebraic constraint in (13) is incompatible with our default set-up for Hopf-bifurcations, see [Uec17b, Uec17a], where (13b) constitutes a constraint at each time t so $s = s(t)$ in (13a). In order to compute Hopf branches[1] we redefine s in (13a) as an average (over one period) speed, and replace the constraint $q(u(t)) = 0$ at each time slice by the averaged constraint

$$q_H(u) := \sum_{i=1}^{m-1} \langle \partial_x u_0, u(t_i) \rangle \stackrel{!}{=} 0, \tag{15}$$

where $t_1, \ldots, t_m$ denotes the gridpoints of the time discretization. Thus, after introducing the unknown period T and rescaling $t = Tt$, i.e., now $t \in [0, 1)$, $t_1 = 0, \ldots, t_m = T$, we consider

$$M\partial_t u = -T(G(u) - s\partial_x u), \tag{16a}$$
$$q_H(u) = 0. \tag{16b}$$

with scalar constraint and $s \in \mathbb{R}$. Hence, (16) is a system of $mn_u + 1$ equations for the $mn_u + 1$ unknowns $(u(t_1), \ldots, u(t_m), s)$, where n_u is the number of unknowns at each time slice.

With ϕ and ψ denoting the phase condition and the arclength condition for Hopf orbits, see [Uec17b], the complete extended system for $U = (u, T, \lambda, s)$ then reads

$$H(U) := \begin{pmatrix} \mathcal{G}(U) \\ \phi(u) \\ \psi(U) \\ q_H(u) \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \in \mathbb{R}^{mn_u+3}, \tag{17}$$

[1] We refer to a branch that bifurcates at a Hopf bifurcation point simply as a 'Hopf branch' and the associated periodic orbits as 'Hopf orbits'.

with Jacobian

$$\mathcal{A}=\begin{pmatrix} \partial_u\mathcal{G} & \partial_T\mathcal{G} & \partial_\lambda\mathcal{G} & \partial_s G \\ \partial_u\phi & 0 & 0 & 0 \\ \xi_{\rm H}\tau_u & (1-\xi_{\rm H})w_T\tau_T & (1-\xi_{\rm H})(1-w_T)\tau_\lambda & w_s\tau_s \\ \partial_u q_H & \partial_T q_H & \partial_\lambda q_H & \partial_s q_H \end{pmatrix}, \tag{18}$$

where $\tau = (\tau_u, \tau_T, \tau_\lambda, \tau_s)$ denotes the tangent along the branch, and ξ_H, w_T, w_s are weights; see [Uec17b, §2.3] for further details and notation, e.g., the definition of $\mathcal{G}$ as a time discretization of $M\partial_t u = -T(G(u) - s\partial_x u$. For (17) we can immediately use a modified Hopf setup for systems with constraints, see [Uec17a, §4 and Appendix A], where two more examples of Hopf bifurcations with symmetries are considered: a reaction diffusion system with mass-conservation, and the Kuramoto-Sivashinsky equation on a periodic domain, which yields a translational and boost symmetry.

We can also compute the Floquet multipliers of (relative) periodic orbits from $\partial_u\mathcal{G}$. Note, however, that since q fixes the space translations, $\partial_u\mathcal{G}$ typically possesses a multiplier close to 1 from space translations of the periodic orbit – in addition to the trivial multiplier 1 from time-translations. As a consequence, the stability computations of periodic orbits tend to be less accurate than for the case without translational invariance, and we slightly relax the tolerance for multipliers close to 1 to be identified as 1, i.e., we work with $\text{tol}_{\rm fl}$ =1e-3 instead of the default setting $\text{tol}_{\rm fl}$=1e-8.

4.1 Modulated fronts: demo `modfro`

Table 3 gives an overview of the scripts and functions in `symtut/modfro` used for (11). In `cmds1` we fix $m = 9$, $l_x = 25$ with an (initial) mesh of $n_x = 51$ points, and choose a as the bifurcation parameter, starting with $a = 0.18$ and initial guess

$$u_0(x) = \frac{1}{2}(\tanh(2x) - 1) \text{ and } v_0 = 1 - u.$$

This is also used as a reference profile in `tintfreeze` (and later on in (15)). For $a = 0.18$ time simulation with `tintfreeze` quickly convergences to a travelling wave, but for, e.g., $a = 0.1$, `tintfreeze` gives a time oscillating near travelling wave profile. Notably, the oscillations in the speed s are rather large, see Fig.7(a), and Listing 15, Cells 1 and 2. Thus, somewhere between $a = 0.18$ and $a = 0.1$ we expect a Hopf bifurcation to a relative periodic orbit, i.e., a modulated front.

Table 3: Main scripts and functions in `symtut/modfro`.

script/function	purpose,remarks
cmds1	main script
modfroinit, oosetfemops	initialization function, and setting of FEM operators.
sG, nodalf, sGjac	rhs G, where the nonlinearity is computed in nodalf, and Jacobian
qf, qfder	the phase conditions $\langle \partial_x u_0, u\rangle = 0$, and its u-derivative
qfh, qfhjac	the Hopf version (15) of the translational phase condition, and its derivative
e2rs	gradient based elements 2 refine selector, used for mesh-refinement in cmds3
oomeshada	adaption of library routine oomeshada to also interpolate the (fixed) reference profile u_0 for the phase condition to the adapted mesh.
tintfreezex	variant of tintfreeze (see §3) which also returns $u(t_j, \cdot)$ for plotting.

```
%% demo modfro, C1: clear and init
close all; format compact; keep pphome; p=[]; lx=25; nx=50; par=[0.2 9 0];
p=modfroinit(p,lx,nx,par,'s1');
%p.u(p.nu+1)=0.1; p1=p;   % change a=par(1) to get an osc.front via tintfreeze
%% C2: use tintfreeze to converge to TW, and plot velocity
nt=1.5e4; pmod=500; vmod=100; dt=0.1; t1=0; vel=[];
[p,t1,vel]=tintfreeze(p,t1,dt,nt,pmod,vel,vmod);
figure(7); clf; h=plot(vel(1,:),vel(2,:)); axis tight; legend('a=0.18');
%% C3: switch on ext. system, cont for 1 step, then  mesh-ref, then cont further
```

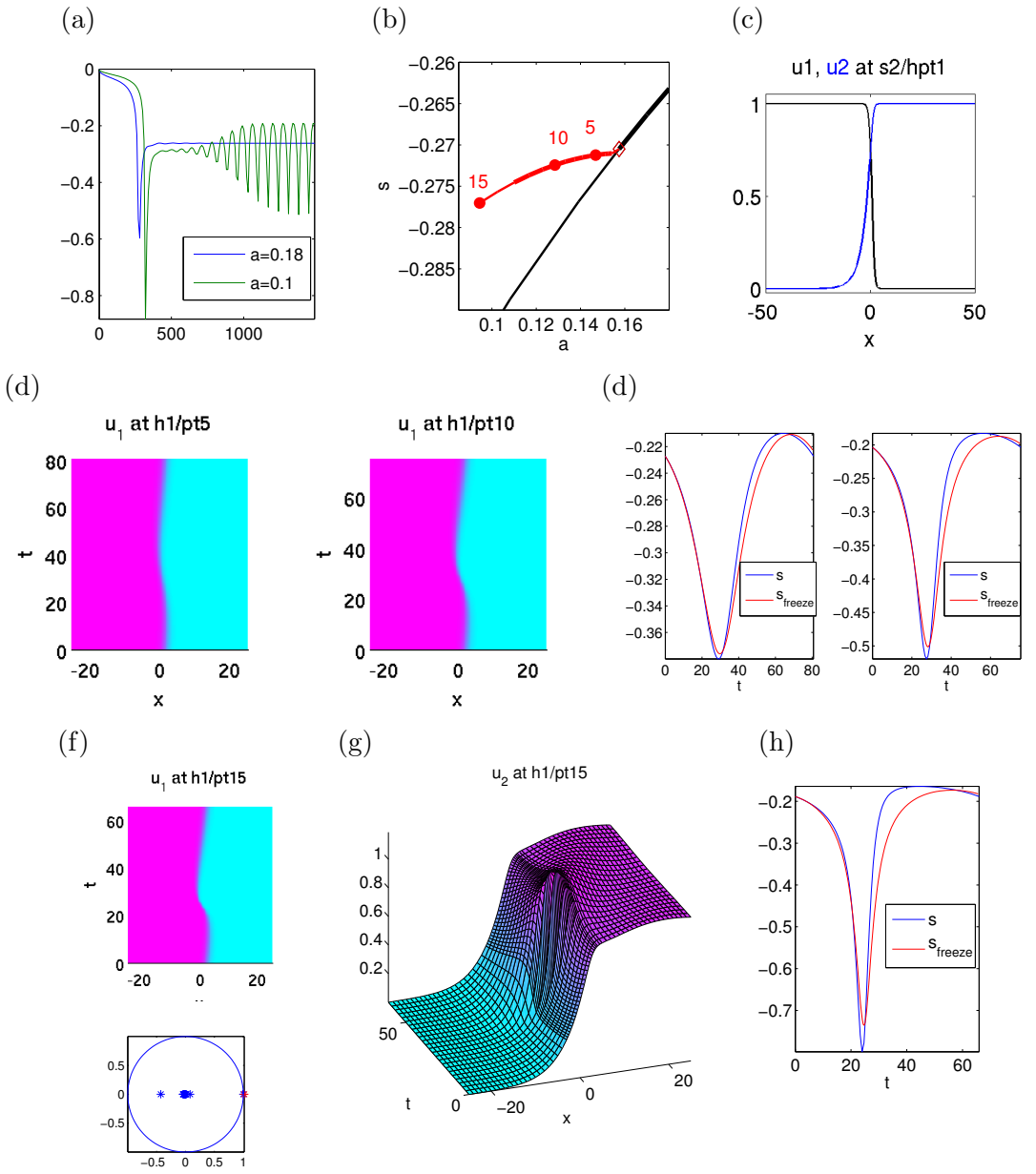

Figure 7: Results from `modfro/cmds1` (Listing 13), with `tl=100` (mesh in t). (a) shows the speed $s(t)$ returned from `tintfreeze` for $a = 0.18$ (convergence to steady TW) and $a = 0.1$ (modulated TW). (b) shows a basic bifurcation diagram. The black branch corresponds to the steady TW with speed=average speed s; at $a \approx 0.157$ there is a Hopf bifurcation for (13) (with profiles in (c)), and the red branch is the bifurcating branch with numerically stable Floquet multiplier until $a \approx 0.1$. Panels (d)-(h) present data from the Hopf branch, at points 5,10 and 15. (d) shows the u profiles of the Hopf orbits u_H at pt5 and pt10 (magenta$\approx$1, cyan$\approx$ 0), and (f) shows the associated speeds obtained from the Hopf orbits (blue) and from `tintfreeze` (red), starting with the same initial condition. Already at $a \approx 0.128$ (pt10), there is a significant error between the two, which would be hard to see in the associated solution plots. This indicates a significant discretization error in u_H: direct simulation with freezing readily allow for a t-mesh fine enough that solutions do not change upon further refinement. (f)-(h) show u_H and $s(t)$ at pt15 ($a \approx 0.094$), together with the Floquet multipliers. (g) illustrates the behavior of the second component v (every 4th point in t is plotted only), and that the adaptive mesh-refinement in space at the front position in Cell 3 of `cmds1.m` makes sense. (h) shows a significant error in the comparison of $s(t)$ and $s_{\text{freeze}}(t)$. The bottom panel of (f) illustrates that there are 2 Floquet multipliers $\mu_{1,2}$ very close to 1, which is always the case along the Hopf branch. At pt15 $\mu_1 \approx 1 + 10^{-5}$, $\mu_2 \approx 1.016$ (red), i.e., numerical instability, which is due to an underresolved t mesh, see text for further comments.

```
p.nc.ilam=[1 3]; p.nc.nq=1; p.u(p.nu+3)=vel(2,end);
p.fuha.qf=@qf; p.fuha.qfder=@qfder; p.sw.qjac=1;
p.sol.ds=-1e-3; p=cont(p,1); p=meshada(p,'ngen',2,'sig',0.005); p=cont(p,10);
%% C4: swibra to Hopf, average speed <s> as aux. variable,
clear aux; aux.nqnew=0; % switch off steady constraint
aux.nqh=1;   % switch on 1 'Hopf' constraint
aux.qfh=@qfh; aux.qfhder=@qfhjac; % function handles to hopf contraints
aux.tw=1e-6; % small weight of period T in arclength often useful
aux.dlam=0; % use trivial initial predictor
aux.tl=100; hodir='h1'; ds=0.1; aux.tl=50; % use 2nd version for quick results
p=hoswibra('s1','hpt1',ds,4,hodir,aux); % branch switching
p.hopf.flcheck=1; p.hopf.fltol=1e-2; % set some additional controls
p.file.smod=1; p.nc.dsmax=ds; p.sw.verb=0;
p.hopf.auxp=@speedplot; % aux function for on the fly plot of s(t)
p.nc.ilam=1; % the primary cont. parameter
p.hopf.ilam=3; % the 2nd active parameter, here again the speed (but now average)
p=cont(p,10);
```

Listing 14: `modfro/cmds1.m` (first 26 lines). Cells 1-3 deal with initialization, convergence to a (steady) TW, and continuation of this in the parameter a, see text. **Cell 4** deals with branch switching to and continuation of the Hopf branch. Most control variables for this continuation are set automatically via `hostanparam` in `hoswibra`, and we only need to pass a few problem specific parameters via the field `aux`. In particular, in line 14 we switch off the (steady) constraint (13b) in the computation of the residual, which thus only considers (13a) at each time time slice. Instead, in lines 15,16 we switch on the 'Hopf constraint' (16b) and set the pertinent function handles, cf. Listings 16 and 17. Additionally, we set a small T weight in the arclength norm, and tell `hoswibra` to use a "trivial" (no normal form computations) initial predictor, i.e., fix the continuation parameter λ in the first predictor. In line 19 we set the time-discretization (rather fine, see text for further comments, and alternatively uncomment the 2nd half) and the output directory for the Hopf orbits. Line 20 contains the actual branch switching. Before continuation in line 26 we set some additional controls and parameters, such as a rather large tolerance for identifying the Floquet multiplier 1 (see text).

```
%% C5: compare Hopf-orbit to tintfreeze: load some point, and freeze again
% OK for tl=100 and a=0.147 (pt5), significant error already at at a=0.128 (pt10)
p=loadp(hodir,'pt5'); hoplot(p,1,1); speedplot(p); ylabel('');
vel=[]; t1=0; t2=p.hopf.T; tl=p.hopf.tl; p.u(1:p.nu)=p.hopf.y(1:p.nu,1);
nt=1000; dt=t2/nt; pmod=round(nt/20); vmod=5;
[p,t1,vel,uv]=tintfreezex(p,t1,dt,nt,pmod,vel,vmod);
figure(10); hold on; plot(vel(1,:),vel(2,:),'r'); axis tight;
set(gca,'FontSize',p.plot.fs); xlabel('t'); legend('s','s_{freeze}');
```

Listing 15: `modfro/cmds1.m` (continued). In **Cell 5** we compare $s(\cdot)$ from the Hopf orbit with $s_{\text{freeze}}(\cdot)$ obtained from `tintfreeze` with $u_H(0,\cdot)$ as an initial condition, see text for further comments. The remainder of `modfro/cmds1.m` deals with plotting.

```
function qf=qfh(p,y) % aux eqns in Hopf, here: sum up shifts wrt u0
qf=0; for i=1:p.hopf.tl; u=y(1:p.nu,i); qf=qf+p.u0x'*(u-p.u0); end
```

Listing 16: `modfro/qfh.m`. Straightforward implementation of the constraint (15).

```
function qfj=qfhjac(p,y) % derivatives of qfh
qfj=zeros(1,p.hopf.tl*p.nu); qfa=0;
for i=1:p.hopf.tl; qfj((i-1)*p.nu+1:i*p.nu)=p.u0x'; end
```

Listing 17: `modfro/qfhjac.m`. Derivative of `qfh`.

In oder to find the Hopf bifurcation of modulated TW, in Cell 3 of `cmds1` we continue the steady branch of (13), with some adaptive mesh-refinement at the front, and find a Hopf bifurcation at $a \approx 0.157$. In Cell 4 we then branch switch to these Hopf orbits, set up as (17), and in Cell 5 the speeds

$$s(t) = \frac{\langle \partial_x u_0, G(u(t)) \rangle}{\langle \partial_x u_0, \partial_x u \rangle},$$

computed a posteriori from the Hopf orbits, are compared with those obtained from freezing, cf. Fig. 7(f,h). In the remainder of `cmds1`, which is not listed, we a posteriori compute Floquet multipliers of Hopf orbits and deal with plotting. The Listing and Figure captions contain some for further details.

Regarding the numerical accuracy of the results from Fig. 7 we remark that the branch of modulated fronts can be continued by alternating `tintfreeze` and decreasing a to $a = 0.05$, say, in a stable manner. In contrast, continuation of the Hopf branch predicts loss of stability as measured by Floquet multipliers around $a \approx 0.11$. This discrepancy is worse for $m = 50$, but relaxes for $m = 200$. Since `tintfreeze` can be run with much smaller `dt` without change of results we attribute the error at small a to underresolved t meshes in the continuation of the Hopf branch. The numerical accuracy of the Floquet multiplier μ_1, theoretically at 1 by time-shift invariance, is on the order of 10^{-5} up to $a = 0.1$, and the (spurious) instability is due to the second multiplier μ_2, associated to spatial translation, which is present in $\partial_u \mathcal{G}$ from (18).

4.2 Breathers: demo `breathe`

We consider (12) on the domain $(-l_x, l_x)$ with $l_x = 50$, with fixed parameters $(\alpha, \beta, \gamma, D) = (0.11, 1, 6, 2)$ and use δ as a bifurcation parameter, starting with $\delta = 0.1$. We focus on the Hopf bifurcation from standing pulses, and thus the continuation of a branch of standing breathers. The computation of a Hopf-bifurcation from traveling pulses to traveling breathers is also possible, but requires rather large domains and fine meshes, and therefore is not included in this demo; see [IIM00] for suitable parameter ranges.

Figure 8 shows some basic results for (12), and for the script file `cmds1.m` and the function files in `symtut/breathe` we directly refer to the source code. The implementations of the phase conditions are as before, and similarly for the implementations of the right hand sides and Jacobians we refer to the sources `sG.m, nodalf.m`, and `sGjac.m`. From the numerical point of view, the three main differences compared to §4.1 are as follows:

1. For (12) we refrain from adaptive mesh refinement in x. This can be done to improve the accuracy of the steady pulses and of the breathing pulses near bifurcation, but ultimately the breathers oscillate over a wide region, so that refinement locally in x is irrelevant.
2. For (12) the Hopf orbits become unstable at small δ, i.e., $\delta = \delta_0 \approx 0.005$. However, in contrast to §4.1, this instability is due to a multiplier crossing the unit circle at -1, and is correct in the sense that also in `tintfreeze` simulations the breather looses stability at δ_0. The time evolution then converges to the spatially uniform solution due to interaction with the boundary. For larger domains we obtain stable widely moving breathers down to smaller δ.
3. For standing pulses and breathers, $s = 0$ with high numerical accuracy, i.e., $|s| \leq 10^{-12}$ for all solutions. Nevertheless, s is required as an active variable for steady and breathing (with s as average speed) pulses, with the associated phase conditions (13b) and (16b), respectively. Without (13b), bifurcations from the pulses cannot be detected in a reliable way, and without introducing (16b) the convergence of Newton loops in the continuation of the breathers fails completely.

References

[BCM99] N. J. Balmforth, R. V. Craster, and S. J. A. Malham. Unsteady fronts in an autocatalytic system. *R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci.*, 455(1984):1401–1433, 1999.

[BGUY17] S. Bier, N. Gavish, H. Uecker, and A. Yochelis. Mean field approach to first and second order phase transitions in ionic liquids, Preprint, 2017.

[BORM14] W.-J. Beyn, D. Otten, and J. Rottmann-Matthes. Stability and computation of dynamic patterns in PDEs. In *Current challenges in stability issues for numerical differential equations*, volume 2082 of *Lecture Notes in Math.*, pages 89–172. Springer, Cham, 2014.

[BT07] W.J. Beyn and V. Thümmler. Phase conditions, symmetries, and pde continuation. In *Numerical continuation methods for dynamical systems*, pages 301–330. Springer, Dordrecht, 2007.

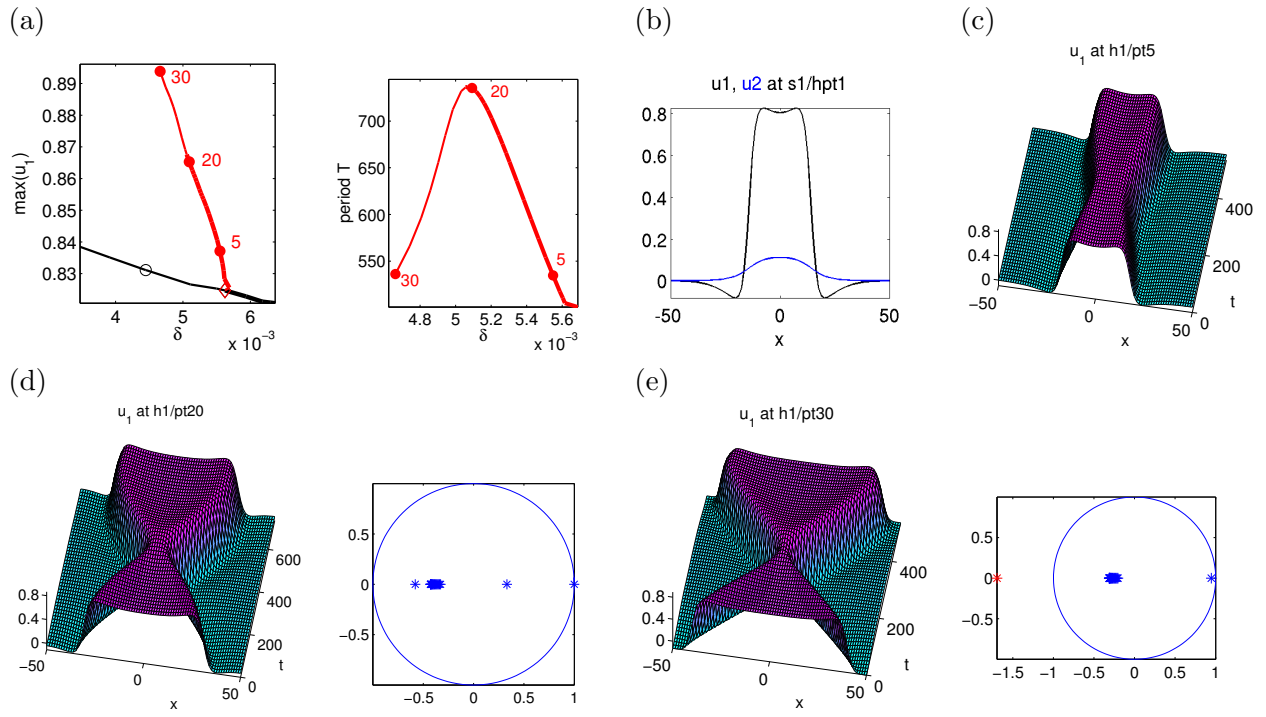

Figure 8: Results from Listing 13, i.e., continuation of (standing) breathers for (12), parameters $(\alpha, \beta, \gamma, D) = (0.11, 1, 6, 2)$ fixed. (a) Basic bifurcation diagram: steady pulse branch (black) and bifurcating breathers (red). (b) u at the bifurcation point. (d)-(f) show narrowly to widely oscillating breathers at points marked in (a), and the Floquet multipliers at pt20 and pt30.

[CBDvHR15] M. Chirilus-Bruckner, A. Doelman, P. van Heijster, and J. D. M. Rademacher. Butterfly Catastrophe for Fronts in a Three-Component Reaction-Diffusion System. *Journal of NonLinear Science*, 25:87–129, February 2015.

[CBvHR12] M. Chirilus-Bruckner, P. van Heijster, and J.D.M. Rademacher. Singularity imbedding and front dynamics of Fitzhugh-Nagumo type systems. *Oberwolfach Reports*, 9(4):3614, 2012.

[DRUW14] T. Dohnal, J. Rademacher, H. Uecker, and D. Wetzel. pde2path 2.0. In H. Ecker, A. Steindl, and S. Jakubek, editors, *ENOC 2014 - Proceedings of 8th European Nonlinear Dynamics Conference, ISBN: 978-3-200-03433-4*, 2014.

[DU17] T. Dohnal and H. Uecker. Periodic boundary conditions in pde2path, 2017.

[GF13] S. V. Gurevich and R. Friedrich. Moving and breathing localized structures in reaction-diffusion systems. *Math. Model. Nat. Phenom.*, 8(5):84–94, 2013.

[GvV14] K. R. Green and L. van Veen. Open-source tools for dynamical analysis of Liley's mean-field cortex model. *J. Comput. Sci.*, 5(3):507–516, 2014.

[HM94] A. Hagberg and E. Meron. Pattern formation in nongradient reaction-diffusion systems: the effects of front bifurcations. *Nonlinearity*, 7(3):805–835, 1994.

[IIM00] T. Ikeda, H. Ikeda, and M. Mimura. Hopf bifurcation of travelling pulses in some bistable reaction-diffusion systems. *Methods Appl. Anal.*, 7(1):165–193, 2000.

[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.

[RU17] J. Rademacher and H. Uecker. The OOPDE setting of pde2path – a tutorial via some Allen-Cahn models, 2017.

[Uec17a] H. Uecker. Hopf bifurcation and time periodic orbits with pde2path – a tutorial, 2017.

[Uec17b] H. Uecker. Hopf bifurcation and time periodic orbits with pde2path – algorithms and applications, Preprint, 2017.

[Uec17c] H. Uecker. `www.staff.uni-oldenburg.de/hannes.uecker/pde2path`, 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.

[Vis07] D. Viswanath. Recurrent motions within plane Couette turbulence. *J. Fluid Mech.*, 580:339–358, 2007.