pde2path with higher order finite elements

Hannes Uecker

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

July 19, 2021

Abstract

We describe by means of some examples how to link the Matlab continuation and bifurcation package pde2path with functions based on the FSElib from [Poz14] implementing a quadratic finite element method (FEM), in 2D and 3D. This quadratic FEM shows some advantages compared to the default piecewise linear FEM of pde2path, in particular with respect to keeping symmetry of solutions. Additionally, in 1D we provide a spectral FEM via Lobatto points on each element.

Contents

1 Introduction

In its standard setup, the Matlab package pde2path [UWR14, Uec19a, Uec21a, Uec21c] uses the piecewise linear (Lagrangian P_1) finite element method (FEM) for numerical continuation and bifurcation analysis of systems of PDEs of the form

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

over bounded domains $\Omega\subset\mathbb{R}^d$, $d=1$, $d=2$, or $d=3$, (1D, 2D, 3D case), with various boundary conditions (BCs). In (1), $u=u(x,t)\in\mathbb{R}^N$, $t\geq 0$, $x\in\Omega$, $\lambda\in\mathbb{R}^p$ is a parameter (vector), $M_d\in\mathbb{R}^{N\times N}$ is a (dynamical) mass matrix, which may be singular, and c, a, b and f may depend on x, λ and u. For (1), pde2path can generate, by a few convenience calls, FEM discretizations based on OOPDE [Prü21], and then assemble the FEM (differentiation) matrices. With these, the user can set up the (discretized) right hand side (rhs) $G(u,\lambda)$, and then compute branches of steady and time periodic solutions. For details, and many demos of numerical continuation of branches of solutions of PDEs, see [Uec21a] and the various tutorials which together with the software and demos can be downloaded at [Uec21c].

Here we explain how to link `pde2path` with higher order FE methods based on the FSElib from [Poz14], and included in the `hofem` folder as `FSElib` and `hofemlib`. As examples we choose three typical model problems, in 1D, 2D and 3D, namely: As a warm up, Allen–Cahn type of equations

$$\partial_t u = c\Delta u + f(u), \quad u = u(x,t) \in \mathbb{R}; \tag{2}$$

as a fourth order problem, the Swift–Hohenberg equation

$$\partial_t u = -(1+\Delta u)^2 + \lambda u + \nu u^2 - u^3, \quad u \in \mathbb{R}; \tag{3}$$

some problems over curved surfaces, for instance the (generalized) Schnakenberg problem

$$\partial_t U = D\Delta_{\mathcal{T}_{R,\rho}} U + F(U),\ F(U) = \begin{pmatrix} -u+u^2 v \\ \lambda - u^2 v \end{pmatrix} + \sigma \left(u - \frac{1}{v}\right)^2 \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \tag{4}$$

on a torus, where $U = \begin{pmatrix} u \\ v \end{pmatrix}$ and $D = \begin{pmatrix} 1 & 0 \\ 0 & d \end{pmatrix}$, and where $\Delta_{\mathcal{T}_{R,\rho}}$ is the Laplace–Beltrami operator (LBO) on the torus. We mainly focus on 2D and 3D, and explain the 1D setting with elements of variable polynomial order up to 5 in Appendix A.

In 2D, the FSElib provides 6–nodes (x_j, y_j), $j = 1, \ldots, 6$, triangle elements (by adding the edge midpoints of a standard triangulation to the discretization) and piecewise quadratic element and test functions, [Poz14, §5.1]. The six element functions $\psi_i(x,y)$, $i = 1, \ldots, 6$ then fulfill as usual

$$\psi_i(x_j, y_j) = \delta_{ij} = \begin{cases} 1 & i = j, \\ 0 & \text{else}, \end{cases} \quad \text{and} \quad \sum_{i=1}^{6} \psi_i \equiv 1, \tag{5}$$

and a function $u : \Omega \to \mathbb{R}$ is expanded as

$$u(x,y) = \sum_{i=1}^{n_p} u_i \psi_i(x,y) \tag{6}$$

where n_p is the total number of points in the mesh, the ψ_i are an enumeration of the global basis functions, and the u_i are the nodal values. In 3D, we have 10–nodes tetrahedra by adding the edge midpoints of a standard triangulation by 4–node tetrahedra to the discretization. The ten quadratic element functions again fulfill

$$\psi_i(x_j, y_j, z_j) = \delta_{ij} \text{ and } \sum_{i=1}^{10} \psi_i \equiv 1, \tag{7}$$

and a function u is expanded as in (6).

For the triangulation, the n_t 6–node triangles in the qFEM thus yield a 3–node triangulation by $4n_t$ 3–node triangles. Similarly, n_t 10–node tetrahedra correspond to $8n_t$ 4–node tetrahedra. As one of the major steps in any FEM is the generation of suitable meshes, we first use the convenience functions for the standard mesh generation in `pde2path`, although the FSElib also includes some functions for mesh–generation on standard domains. Focusing for now on the 2D case, we can subsequently convert a 3–node triangulation to a 6–node triangulation, which we use to assemble the associated mass and stiffness matrices M_6, K_6. The 6–node triangulation also yields a refined 3–node triangulation which we use for plotting, and to generate FEM matrices for which we do not yet have 6–node implementations,

for instance matrices to encode boundary conditions (BCs). In 2D, we mainly use the functions

$$\texttt{p=tri2six(p)} \text{ and } \texttt{[K,M]=assem6g(p,c,a)}, \tag{8}$$

where `p` is a standard `pde2path` problem struct, containing the mesh points in `p.pdeo.grid.p`, and data for the `hofemlib` in `hofem`. This is the 6–node triangulation `hofem.tri`, which is generated by `p=tri2six(p)`, and some switches, see Table 1. In (8), `c` and `a` contain the diffusion tensor c and the matrix a from (1), see Remark 4.1 for details, and there is the simplified version `[K,M]=assem(p)` with $c, a = 1$. By (8) we assemble the 6–node stiffness matrix K for the diffusion operator ("Laplacian") $-\nabla \cdot (c\nabla u)$ associated to c, with homogeneous Neumann BCs, and the 6–node mass matrix M associated to a. Otherwise we completely rely on the original `pde2path` setup for all other assembly and mesh handling. The analogs of this also apply in 3D. For (2) and (3) with (homogeneous) Neumann BCs (NBCs), we thus have a genuine quadratic FEM, but, e.g., for other BCs or for convection matrices to encode phase conditions we have a 'hybrid' FEM. See below for further comments. Table 2 lists the main functions from `hofem/hofem` for using the FSElib; a basic selection of functions from the FSElib is in `hofem/FSElib`, but we strongly recommend [Poz14] and `http://dehesa.freeshell.org/FSELIB/` for further information.

Table 1: Data in `p.hofem`, where `p` is a standard `pde2path` problem struct; at bottom: 1D data.

field	remarks
tri	nt×6 field of 6–node triangle point indices; nt×10 for 10–node tetrahedra
Kfn	filename for saving K,M (as assembly may become slow for larger scale problems)
sw, isw	convenience flag to signal the use of `hofem`, and switch to choose the interpolation method for assembly
t2sinterpolsw	convenience flag for `tri2six` and `four2ten` to interpolate the current solution and tangent to the new mesh (if 1) or not (otherwise).
ne,xe	# of elements, and their endpoints xe(1:ne+1); the meshpoints are still in p.pdeo.grid.p
femorder	(uniform) order, i.e., femorder+1 meshpoints/ansatz functions in each element
npoly	vector of orders for elements; can be set individually, but for mesh–adaption uniform order is assumed
tri	ne×(femorder+1) vector of element nodes

As usual in `pde2path`, the rhs (and Jacobians) for semilinear problems such as (2)–(4) will be encoded in functions `sG` (and `sGjac`). Thus, the main changes compared to the standard treatment of (2) and (3) on various domains and with various BCs via the linear FEM is that at init we need to set up a 6-node (2D) resp. 10-node (3D) triangulation of the spatial domain, and then compute the (6-node or 10–node) stiffness/mass matrices used in `sG` and `sGjac`, and similar in 1D.

To use the `hofemlib`, call `sethofem` in the root directory `hofem`, which also contains the demos coming with this tutorial, see Table 3. As indicated, the demos consider problems treated before. For `ac1D` (Appendix A) and `ac2D`, there is not much benefit from using the qFEM, and they merely serve as toy problems to illustrate the new setup. For the larger scale problems `sh2D` and `sh3D`, an *important advantage* of the qFEM is that it more robustly keeps symmetries of solution branches, i.e., the continuation is less prone to "branch jumping", by which we mean the undesired switching to another branch, often associated with a loss of symmetry, see [Uec21a, §3.6.2]. This also holds for `schnaktor` and the other demos dealing with problems on surfaces. The main purpose of `schnaktor` thus is to give another example how to seamlessly link `hofemlib` with the standard linear FEM setup. The main shortcomings of our qFEM currently are: For BCs other than homogeneous NBCs, which hence require book keeping of boundary segments, we are presently restricted to somewhat simple domains, where boundary identification numbers (boundary IDs) can be set in an easy way.

Table 2: Main functions from `hofemlib`, where `p` is a standard `pde2path` problem struct.

function	remarks
`p=tri2six(p)`	convert the 3-node triangulation from `p.pdeo.grid` into a 6-node triangulation by adding the edges midpoints, storing the 6–node connectivity in `p.tri`. If `p.hofem.t2sinterpolsw=1`, then the current solution u and tangent τ are interpolated to the new mesh.
`[K,M]=assem6g(p,c,a)`	assemble the 6–node triangles Neumann Laplacian stiffness matrix `K`, and the mass matrix `M`. Different options to pass the diffusion tensor `c` (scalar, or 2×2, possibly x–dependent), and a (scalar, possibly x-dependent).
`[K,M]=assem6(p)`	simplified (and slightly faster) version of `assem6g` with $c = a = 1$.
`p=four2ten(p)`	analog of `tri2six` in 3D
`[K,M]=assem10g(p,c,a)`	analog of `assem6g` in 3D. `assem10` as simplified version.
p=two2lob(p)	1D; generate inner (lobatto) nodes for each element, with endpoints p.hofem.xe(1:ne), and the associated connectivity matrix `p.hofem.tri`; store the points in `p.pdeo.grid.p`.
`[K,M]=assem1Dlob(p)`	assemble K,M for Lobatto-FEM, data in p.hofem.
`[K,M,Kx]=assem1Dq(p)`	assemble K,M, and Kx corresponding to ∂_x for quadratic-FEM, data in p.hofem.
`p=simplecoarselob(p)`	for 1D mesh–adaptation.

Moreover, a major strength of the FEM is adaptive mesh refinement. This is easy and flexible in 1D (see §A), but is not yet implemented in the (in our) qFEM in 2D and 3D. However, as a workaround we can do mesh adaptation in the linear FEM, and then switch back to the qFEM.

Remark 1.1 Naturally, we may as well aim to link `pde2path` with other FEM packages. The so called PTE (points-triangulation-edges) data structures for these follow some standards, with minor variations. However, the overheads to assemble mass and stiffness matrices differ significantly from package to package, and in particular when aiming at higher order FEM and easy `Matlab` interfaces, the choices of packages do not seem great. See also [FALD12].

If the nodes in each element (e.g., interval, triangle or tetrahedron) are chosen to minimize the interpolation error in each element, e.g., Chebychev or Lobatto points, then such higher order elements are also called spectral elements. See, e.g., [Lui11, Chapter 3] and [Poz14] for very readable and useful introductions. Here we restrict to the functions from the FSElib [Poz14], with minor extensions, and regard this as a first step towards higher order FEM in `pde2path`. ⌋

2 The basic setup, and ac2D

In `ac2D` we consider the (cubic–quintic) AC equation

$$G(u) := -c\Delta u - \lambda u - u^3 + u^5 \stackrel{!}{=} 0 \tag{9}$$

over $\Omega = (-2\pi, 2\pi) \times (-\pi, \pi)$, with the Dirichlet BCs (DBCs)

$$u = d\cos(y/2) \text{ on } \{x = 2\pi\}, \text{ parameter } d, \tag{10}$$

and $u = 0$ on the remaining boundary. For $d = 0$ we have the bifurcation points

$$\lambda_{jl} = (j/4)^2 + (l/2)^2, \quad \phi_{jl} = \sin\left(\frac{j}{4}(x+2\pi)\right)\sin\left(\frac{l}{2}(y+\pi)\right), \quad j,l = 1,2,\ldots \tag{11}$$

Table 3: Subdirectories of `hofem`; the last two demos are not discussed in detail here.

directory	remarks
`FSElib`	Selected functions from [Poz14], see also `http://dehesa.freeshell.org/FSELIB/`
`hofemlib`	main library, minor modifications of functions/code snippets from [Poz14]
`ac1D`, `ac1Dq`	(2) over a 1D interval with Dirichlet BCs, similar to `demos/acsuite/ac1D`; see Appendix A.
`ac2D`	(2) over a 2D box with Dirichlet BCs, similar to `demos/acsuite/ac2D`, see also [Uec21a, §3.1]
`sh2D`	(3) on a (rather large) 2D rectangle with NBCs, focusing on the computation of a primary hexagon branch and a snaking branch of fronts between hexagons and $u = 0$. This is similar to [Uec21a, §8.2.3], where analogous problems have been treated with the linear FEM, see also `demos/pftut/sh`. Moreover, the same problem has been considered in [Uec21b, §5.2.2] using FFT methods. We comment on comparison of the different discretization methods below.
`sh3D`	(3) in 3D with NBCs over a long and slender bar, similar to [Uec21a, §8.3.2], again see also `demos/pftut/sh`.
schnaktor	(4), using `assem6g` to build the LBO in 6–node triangles, with periodic BCs and hence some phase–condition matrices. Similar to [Uec21a, §10.2.2].
acS	(2) on a sphere as another canonical example, similar to [Uec21a, §10.1]
schnakcone	(4) on a cone, hence with a non–diagonal LBO, similar to [Uec21a, §10.2.3].

from the trivial branch $u \equiv 0$, which we can use to assess accuracy. The same problem is studied in the demo `demos/acsuite/ac2D` via linear FEM, see also [RU19] and [Uec21a, §6.3], where it also serves as a model problem for mesh adaptation, and to which we refer for details.

Here we want to explain the usage of the qFEM (6–node triangles), and give just one brief comparison of accuracy based on (11). Listing 1 shows the first 8 lines of `acinit.m`. In l4 we generate a (3–node triangle) mesh for the domain as usual, and the essential new command is `tri2six` in line 5. This adds the midpoints of the given triangle edges to the point-list, stores the associated 6–node triangles in `p.hofem.tri`, and the new (3–node triangles) mesh in `p.pdeo.grid`. Subsequently we also set the boundary IDs via `setidssq`, which assigns 1=bottom, 2=right, 3=top, 4=left.

```
function p=acinit(p,lx,ly,nx,par,hofem) % ac2D
p=stanparam(p); screenlayout(p); p.nc.neq=1; p.sw.sfem=-1; p.hofem=hofem;
p.fuha.sG=@sG; p.fuha.sGjac=@sGjac; p.fuha.e2rs=@e2rs; p.nc.sf=1e3;
pde=stanpdeo2D(lx,ly,2*lx/nx); % 2D rectangle pde object, h as argument
p.pdeo=pde; p=tri2six(p); % convert 3-node triang.to 6-node-triang:
% add edge-midpoints, store 6-node-elems in p.hofem.tri, mesh in p.pdeo.grid
p.pdeo.grid=setidssq(p.pdeo.grid); % set ids for boundary segments
%for i=1:4; identifyBoundarySegment(p.pdeo.grid,i); pause; end
```

Listing 1: Start of `ac2D/acinit.m`. Remainder as usual.

In `oosetfemops` in Listing 2 we use `p.hofem.sw` to assemble either the 6-node or the 3-node K and M, while the boundary matrices Q and G^{BC} are always assembled by the standard 3–node functions. The rhs `sG` and the Jacobian `sGjac` then take the same form as usual.

```
function p=oosetfemops(p) % for ac2D, in linear (P1) or quadatic (P2) FEM
gr=p.pdeo.grid; fem=p.pdeo.fem; % just shorthands
if p.hofem.sw; [K,M]=assem6(p); % use 6-node triangulation, simple version c=a=1
else [K,M,~]=fem.assema(gr,1,1,1); end % or default 3-node
p.mat.K=K; p.mat.M=M; % store K and M
bc1=gr.robinBC(1,0); bc2=gr.robinBC(1,'cos(y/2)'); % BC matrices based on 3-node
gr.makeBoundaryMatrix(bc1,bc2,bc1,bc1); % bottom, right, top, left
[p.mat.Q,p.mat.G,~,~]=p.pdeo.fem.assemb(gr); % the BC matrices
```

```
function r=sG(p,u)  % ac2D,
f=nodalf(p,u);  % the nonlinearity
par=u(p.nu+1:end); u=u(1:p.nu); % split u into parameters and PDE-part
r=par(1)*p.mat.K*u-p.mat.M*f... % the bulk part
```

```
+p.nc.sf*(p.mat.Q*u-par(4)*p.mat.G); % the boundary terms via stiff-spring
```

Listing 2: `ac2D/oosetfemops.m` and `sG`. In `oosetfemops` we use the switch `p.hofem.sw` to either use the 6–node triangulation `p.tri` or the default 3–node triangulation to assemble K and M, with point coordinates for both in `p.pdeo.grid.p`.

With these preparations, the script `ac2D/cmds1.m` works almost identical to the one in `acsuite/ac2D`. One important difference is that we run the initialization

```
lx=2*pi; ly=pi; nx=20; sw6=0; p=acinit(p,lx,ly,nx,par,sw6)
```

with a rather small n_x, since the generation of the 6–node triangulation (whether used or not, according to the switch `p.hofem.sw`) at least doubles the number of mesh points. In detail, we always generate the finer mesh via `tri2six`, and `p.hofem.sw` only decides whether the associated 6-node mesh or the standard 3–node triangulation from `p.pdeo.grid.t` are used for assembly in `oosetfemops`.

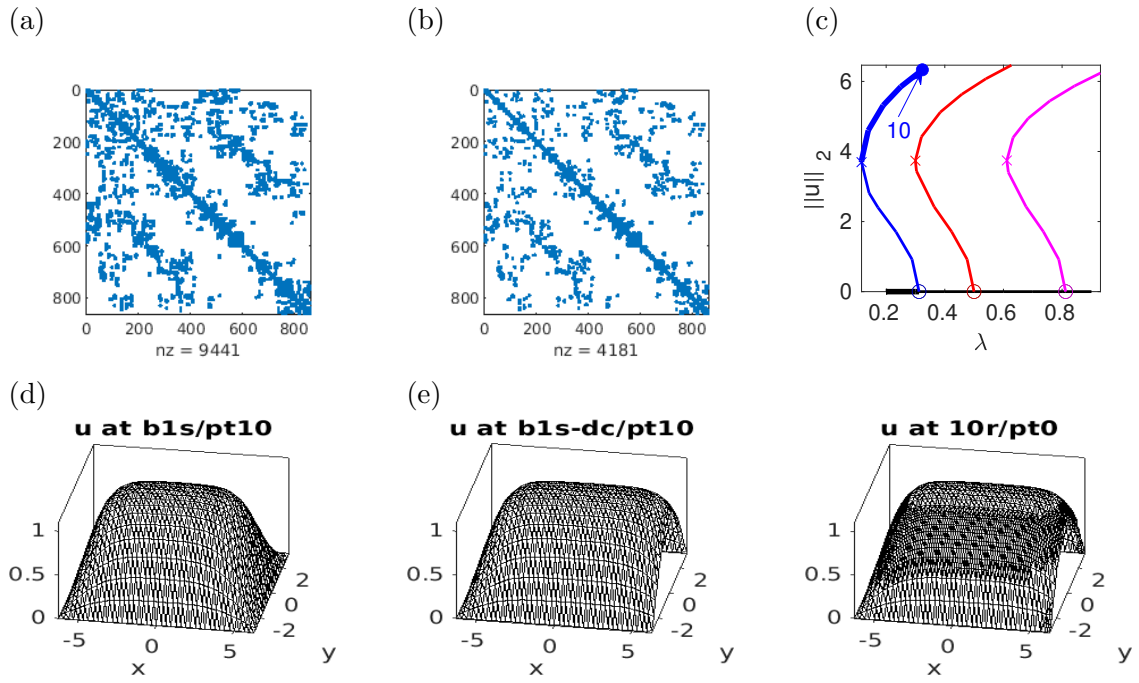

Figure 1: (2) over $\Omega = (-2\pi, 2\pi) \times (-\pi, \pi)$ with the BCs (10). (a) sparsity structure of K for a triangulation with $n_p =$ points and $n_t =$ 6-node triangles. (b) K for the corresponding 3–node triangles. (c) basic BD for $d = 0$. (d,e) sample solutions, including mesh adaptation.

Figure 1 shows some basic results from `cmds1`. In (a),(b) we compare the sparsity structures of the stiffness matrices K_6 (6-node triangles) and K_3 (same mesh points, but linear FEM, i.e., 3–node triangles). As expected K_3 is roughly twice as sparse as K_6, and the same sparsity structures hold for M_6 and M_3. To assess the 6–node vs 3–node accuracy, at the start of `cmds1` we compute $e_3^2 := \sum_{i=1}^3 (\lambda_i - \lambda_{i,3})^2$ and $e_6^2 := \sum_{i=1}^3 (\lambda_i - \lambda_{i,6})^2$, where $(\lambda_1, \lambda_2, \lambda_3) = (5/16, 1/2, 13/16)$ are the first three analytical BPs, and $\lambda_{i,3}$ and $\lambda_{i,6}$ are their numerical approximations based on 3–node triangles and 6–node triangles, respectively. This yields $e_3 \approx 0.008$ and $e_6 = 0.004$, indicating the higher accuracy of 6–node triangles, *on the same mesh*, which balances the slightly higher costs due to the denser matrices.

In (c) we show a basic BD of the first 3 nontrivial branches, on a rather coarse mesh of $n_p = 861$ points (400 6–node triangles), and a sample solution in (d), while (e) shows a sample solution from a continuation in the boundary amplitude d, and an example of mesh adaptation from $n_p = 861$ to $n_p = 1886$. This works by setting `p.hofem.sw=0` and hence using only the 3–node assembly.

Subsequently, we can again use `tri2six` to extend the refined 3–node mesh to a 6–node mesh (with $n_p = 7406$ mesh points), and switch back to using the 6–node triangulation by setting `p.hofem.sw=1` and calling `oosetfemops` (see source of `cmds1.m`). A drawback of this workaround for adaptive mesh refinement in the 6–node setting is that it quickly leads to excessively many mesh points. However, this is also due to here using the standard FEM error–estimator and refinement without coarsening; see §3 for the alternative `trullekrul` mesh adaptation.

3 Swift–Hohenberg

The fourth order (quadratic-cubic) SH equation (3), i.e.,

$$\partial_t u = -(1+\Delta u)^2 + \lambda u + \nu u^2 - u^3, \quad u \in \mathbb{R}, \tag{12}$$

is a canonical model for pattern formation, see, e.g., [SU17] or [Uec21a, Chapter 8] and the references therein. We consider (12) over 2D and 3D boxes (rectangles and cuboids, respectively), with NBCs for u and Δu. The trivial solution branch $u \equiv 0$ is stable for $\lambda < 0$, and depending on the domain Ω (in particular its aspect ratio(s)), near $\lambda = 0$ becomes unstable to patterns with wave length near 2π, i.e., with wave vectors $k = (k_1, k_2)$ or $k = (k_1, k_2, k_3)$ with $|k|^2 = k_1^2 + \ldots + k_d^2 \approx 1$. Letting $(u_1, u_2) = (u, \Delta u)$ we obtain the 2nd order system

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \partial_t \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} -\Delta u_2 - 2u_2 - (1-\lambda)u_1 + f(u_1) \\ -\Delta u_1 + u_2 \end{pmatrix}, \tag{13}$$

which immediately translates into the FEM formulation

$$\mathcal{M}\dot{u} = -(\mathcal{K}u - F(u)), \tag{14}$$
$$\mathcal{M} = \begin{pmatrix} M & 0 \\ 0 & 0 \end{pmatrix},\ \mathcal{K} = \begin{pmatrix} 0 & -K \\ K & M \end{pmatrix},\ F(u) = \begin{pmatrix} M((\lambda-1)u_1 - 2u_2 + f(u_1)) \\ 0 \end{pmatrix},$$

where K and M correspond to the scalar stiffness and mass matrices. See, e.g., [Uec21a, Remark 8.1] for the equivalence of (13) and (12) for domains with a smooth boundary, or convex polygonal domains.

3.1 2D

In 2D we choose Ω a rectangle with side lengths $l_x = 12\pi$ and $l_y = l_x/\sqrt{3}$, which corresponds to the three critical wave vectors $k_1 = (1, 0)$, $k_{2,3} = (-1/2, \pm\sqrt{3}/2)$, forming a hexagonal dual lattice. Correspondingly, the primary bifurcations at $\lambda = 0$ are to

$$\begin{aligned} &\text{stripes } u = \pm 2\sqrt{\lambda/3}\cos(x) + \mathcal{O}(\lambda^{3/2}) \text{ and} \\ &\text{hexagons } u = A\big[\cos(x) + \cos((x+\sqrt{3}y)/2) + \cos((x-\sqrt{3}y)/2\big] + \mathcal{O}(\lambda^2)), \end{aligned}$$

with $A = -\gamma\lambda + \mathcal{O}(\lambda^2)$, $\gamma = 2\nu + \mathcal{O}(|\lambda\nu|)$. The stripes always bifurcate in pitchforks, and the hexagons bifurcate transcritically for $\nu \neq 0$. The subcritical solutions (with $A > 0$) are called up hexagons (or spots), the supercritical solutions are called down hexagons (or gaps). The up hexagons stabilize in a fold at $\lambda = -\frac{\nu^2}{18} + \text{h.o.t}$, see [Uec21a, §8.1], and due to this we may expect snaking branches of localized up hexagons (or fronts between hexagons and $u = 0$). However, in particular over large domains these localized hexagon branches are not easy to continue numerically in the piecewise linear

FEM discretization, because they (and other branches) are prone to "branch jumping" due to the large multiplicity of patterns. Hence, continuing these branches can be thought as a benchmark problem.

For the linear FEM in [Uec21a, §8.2.3] we need to carefully choose sufficiently fine and symmetric meshes, and, moreover, need to use the modification `pmcont` to mitigate the branch jumping. In [Uec21b] we studied the problem using spectral discretizations by FFT, finding that this yields quite robust continuations with moderately fine discretizations (n_p around 5000), but the drawback is that the FFT discretization yields full Jacobians, and adaptive mesh refinement is not possible. Here we find that 6–node triangulations also allow more robust continuations (no branch jumping) with again rather coarse discretizations.

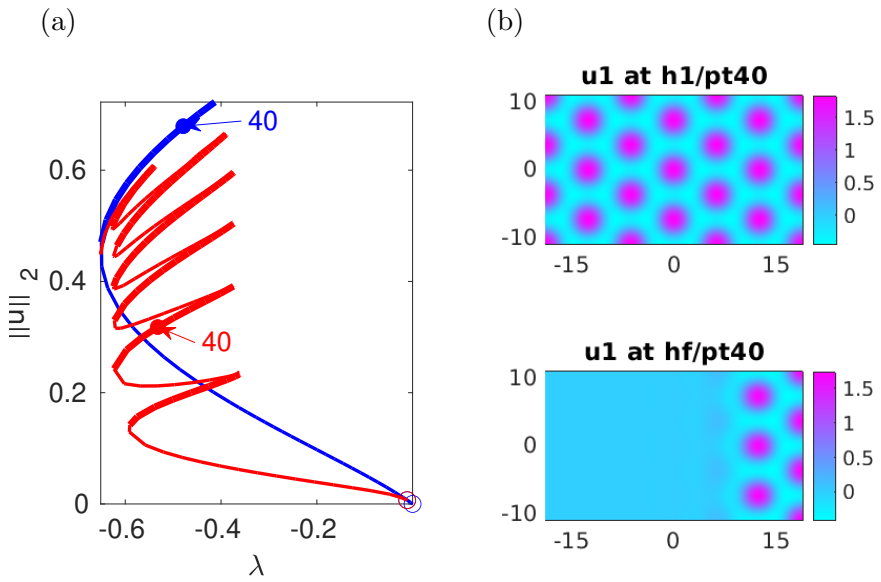

Figure 2: (12) on $\Omega = (-l_x, l_x) \times (-l_y, l_y)$, $l_x = 6\pi$, $l_y = l_x/\sqrt{3}$, $\nu = 2$, 6-node triangulation with $n_p = 5275$. (a) BD of hexagons (blue) and hexagon fronts (red). (b) sample solutions.

In Fig. 2 we show the basic BD of the up hexagons and the hex–to–zero front, on a 6–node triangle mesh with $n_p = 5275$. In Fig. 3 we illustrate mesh adaptation using `trullekrul` [Uec19b] on `hf/pt40`, cf. also [Uec21a, §8.2.3]. The `trullekrul` mesh adaptation, based on 3–node triangles, includes genuine coarsening, which on `hf/pt40` naturally yields coarse meshes for negative x where u is almost identically zero; see (b) for a sample, where the degree of coarsening strongly depends on the used `trullekrul` parameters. Thus, `trullekrul` gives more options to control the meshing, although, as indicated in (c), switching back to 6–node triangles by the naive method of subdivision at the edge midpoints naturally increases n_p again.

As we do not save `p.mat` to disk[1] and as `assem6` (and more so `assem10` for 3D) due to a lack of vectorization becomes somewhat slow for $n_p > 5000$, say, in `oosetfemops` we apply a little trick to avoid reassembling of K, M when reloading points for, e.g., branch–switching: we first try to load K, M from the file `p.hofem.Kfn`; only if this fails, then K, M are assembled, and saved to file `p.hofem.Kfn`. See Listing 3.

```
p=[]; dsw=1; nref=2; lx=6*pi; nx=35; ly=lx/sqrt(3); ny=round(ly/lx*nx);
Kfn='K1.mat';try delete(Kfn); end; hofem.Kfn=Kfn; hofem.sw=1; % hofem data
lam=-0.01; nu=2; par=[lam nu]; p=shinit(p,lx,ly,nx,ny,dsw,par,nref,hofem);
```

```
function p=oosetfemops(p) % for SH as 2nd order system, hence singular p.mat.M
if p.hofem.sw;  % 6-node, simple syntax (c=a=1) [K,M]=assem6(p);
```

[1] to save disk space; in fact, the K and M for $n_p = 10000$ already require about 1MB, and this can quickly increase to 10 or 20 MB for finer meshes

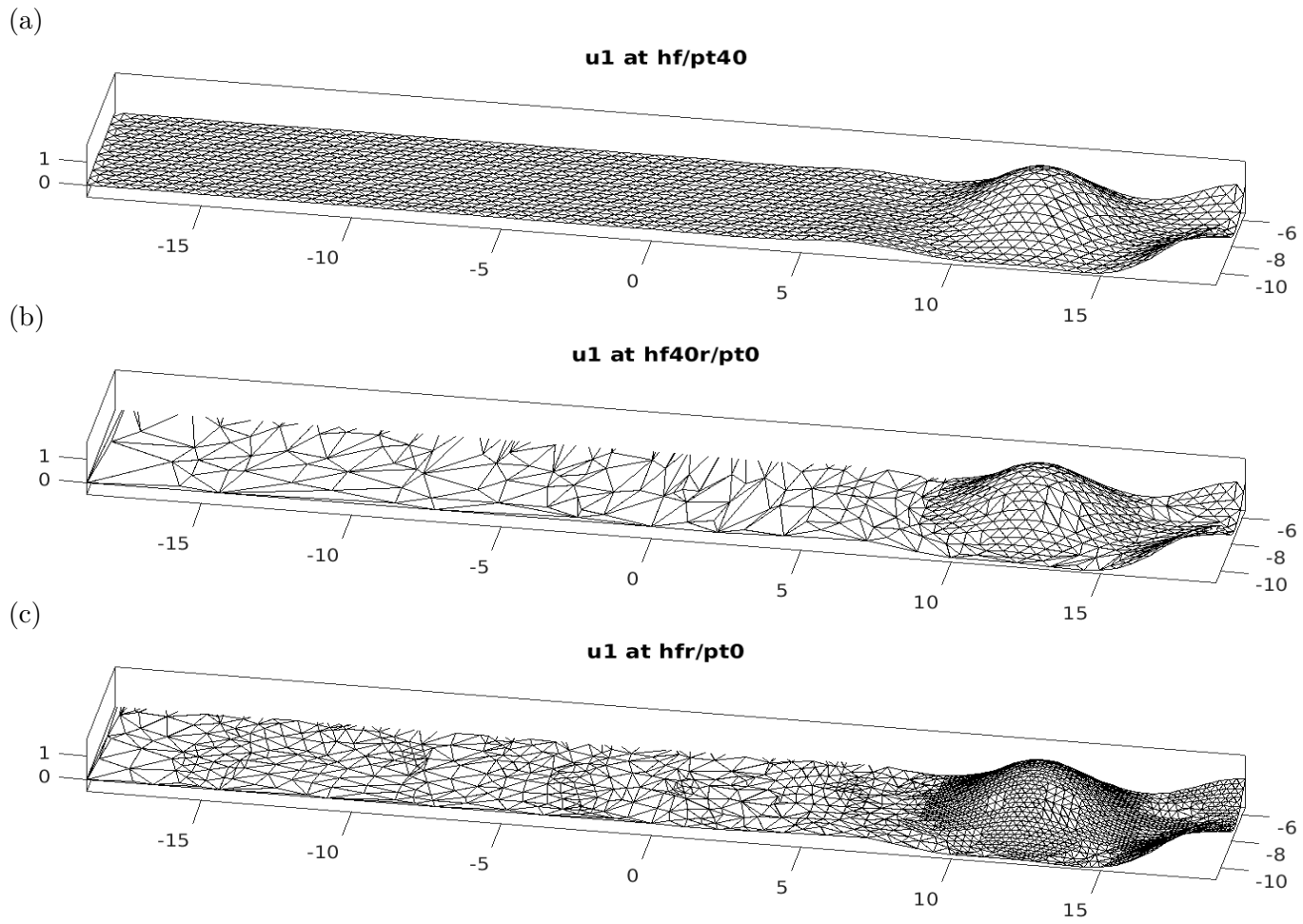

Figure 3: Solution plots for mesh adaptation for `hf/pt40` from Fig. 2. (a) original mesh, $n_5 = 5275$. (b) after `trullekrul` refinement/coarsening, $n_p = 1644$. (c) returning to 6-node triangles, $n_p = 6494$.

```
 try K=load(p.hofem.Kfn,'KM'); K=K.KM.K; M=K.KM.M; % try loading K,M from disk
% if that fails, assemble K,M and save
 catch;tic; [K,M]=assem6(p); toc, KM.K=K; KM.M=M; save(p.hofem.Kfn,'KM'); end
else [K,M,~]=p.pdeo.fem.assema(p.pdeo.grid,1,1,1); % standard 3-node
end
p.mat.M=[[M 0*M];[0*M 0*M]];  % system mass matrix (here singular)
p.mat.Ks=K; p.mat.Ms=M;   % save SCALAR Laplacian, system K composed in sG
```

Listing 3: `sh2D6/cmds.m` (lines 3-5) and `oosetfemops.m`, where we first try to load K and M from file `p.hofem.Kfn`, and only if this fails compute (and save) K and M.

3.2 3D

The basic ideas for discretization of a 3D domain Ω by 10–node tetrahedra [Poz14, §8.7] are similar to those for 6–node triangles in 2D. We start with a standard `pde2path` discretization of Ω by 4–node tetrahedra, for instance as generated by `pdeo=stanpdeo3D(lx,ly,lz,h)`, containing the point–coordinates in `pdeo.grid.p` $\in \mathbb{R}^{3\times n_p}$ and the point–indices of the tetrahedra in `pdeo.grid.t` $\in \mathbb{R}^{5\times n_t}$, where the 5th row is a subdomain identifier. Then, using `p=four2ten(p)` we generate a 10–node discretization by adding the 6 edge midpoints for each tetrahedron, again modifying `p.pdeo.grid` accordingly, and storing the 10-node tetrahedra list in `p.hofem.tri`. The files in `sh3D10` thus are completely analogous to those in `sh2D6`, and we only briefly discuss the results obtained in `cmds1` (small domain) and `cmds2` (long domain).

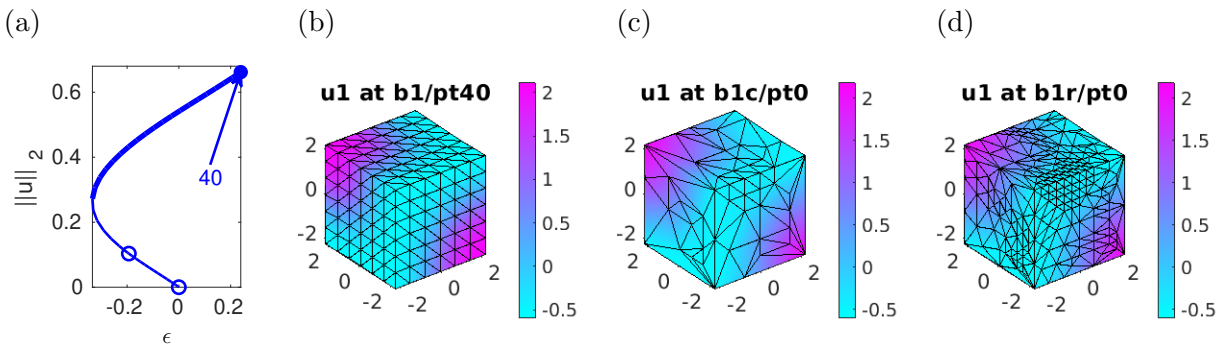

Figure 4: Results from `sh3D10/cmds1.m`. (a) BD of BCCs on $\Omega = (-l_x, l_x)^3$, $l_x = \pi/\sqrt{2}$, $n_p = 343$. (b) sample solution. (c,d) sample solution after coarsening to $n_p = 205$, and after `four2ten` with $n_p = 1403$. Again the mesh after coarsening strongly depends on the chosen trullekrul parameters; see `cmds1.m`.

In Fig. 4(a) we start with a BD of the primary body centered cubic (BCC) balls branch bifurcating from $\lambda = 0$, on a minimal domain $\Omega = (-l_x, l_x)^3$, $l_x = \pi/\sqrt{2}$, and a coarse mesh with $n_p = 343$, with a sample solution in (b). In (c) we illustrate results of mesh coarsening on the 4–node tetrahedra level via trullekrul (c), and subsequent return to 10–node tetrahedra in (d).

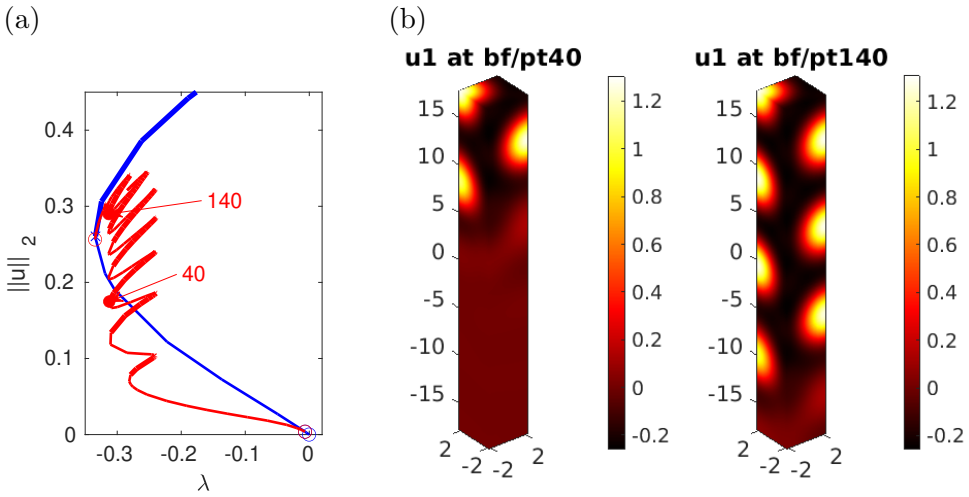

Figure 5: Results from `sh3D10/cmds2.m`. (a) BD of hot balls (blue) and a balls–to–zero fronts (red) for $\Omega = (-l_x, l_x) \times (-l_y, l_y) \times (-l_z, l_z)$, $l_x = l_y = \pi/\sqrt{2}$ and $l_z = 8l_x$, 10–node tetrahedra with $n_p = 11495$. (b) sample solutions.

In Fig. 5 we consider a 3D analog of Fig. 2, namely the BCC balls on a slender bar $\Omega = (-l_x, l_x) \times (-l_y, l_y) \times (-l_z, l_z)$ with $l_x = l_y = \pi/\sqrt{2}$ and $l_z = 8l_x$. This is discretized by $n_p = 11495$ mesh points, which yields nz=293475 non–zero entries in K_{10}. Like the up hexagons in Fig. 2, the balls bifurcate transcritically, and the subcritical blue part stabilizes in a fold. Additionally there are BPs on the blue branch to localized balls, and branch switching at the first BP yields the snaking red branch of steady fronts between balls and $u \equiv 0$. At the end of `cmds2.m` we illustrate `trullekrul`-mesh adaptation on `bf/pt40`, which works just like in Fig. 4.

4 Problems on curved surfaces

To illustrate how to work with more general diffusion operators, and phase conditions, in `schnaktor` we consider the reaction diffusion system (4), i.e.,

$$\partial_t U = D\Delta_{\mathcal{T}_{R,\rho}} U + F(U),\ \ F(U) = \begin{pmatrix} -u+u^2v \\ \lambda - u^2 v \end{pmatrix} + \sigma \left(u - \frac{1}{v}\right)^2 \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \tag{15}$$

on the (surface of the) torus with major radius $R > 0$ and minor radius $0 < \rho < R$, parameterized by

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} = \begin{pmatrix} (R + \rho\cos y)\cos x \\ (R + \rho\cos y)\sin x \\ \rho \sin y \end{pmatrix} \in \mathbb{R}^3, \quad (x, y) \in \Omega = [-\pi, \pi)^2. \tag{16}$$

The associated Laplace–Beltrami operator (LBO) is denoted by $\Delta_{\mathcal{T}_{R,\rho}}$ and given by

$$\Delta_{\mathcal{T}_{R,\rho}} u(x, y) = \frac{1}{\rho^2(R + \rho\cos y)} \partial_y((R - \rho\cos y)\partial_y u) + \frac{1}{(R + \rho\cos y)^2}\partial_x^2 u, \tag{17}$$

with periodic BCs in x (think azimuth ϕ) and y (think elevation θ). The problem has been considered in [Uec21a, §10.2.2] in the linear FEM, and our main task is to set up (17) using `assem6g`, and the x derivative matrix `Dphi` needed for a phase condition to deal with the translational invariance in x. Listing 7 shows the pertinent functions. In `schnaktorinit`, the main difference to `shinit` from before is the switching–on of the periodic BCs in line 14; this works as usual [Uec21a, §4.3] via `p=box2per(p,psw)`, where (here) `psw=[1 2]` means pBCs in x and y. In `oosetfemops`, similar to the BC matrix `Q` in `ac2D`, `Dphi` is assembled using the 3–node triangulation, and `LBtor6` works exactly like the 3–node version `LBtor`, except that `assem6g` is used instead of `assema`. Finally, the script `cmds1.m` works exactly like the 3–node version from [Uec21a, §10.2.2] and `demos/pftut/schnaktor`. The results as shown in Fig. 6 are also the same, but again the 6–node triangles are more robust wrt keeping symmetry, although this only becomes an issues on larger tori or at (significantly) lower λ.

```
function p=oosetfemops(p) % Schnakenberg on torus,
gr=p.pdeo.grid; fem=p.pdeo.fem; par=p.u(p.nu+1:end); u=p.u(1:p.nu);
R=par(4); rho=par(5);
if p.hofem.sw % quadratic FEM
 try KM=load(p.hofem.Kfn,'KM'); K=KM.KM.K; M=KM.KM.M; % try get K,M from disk
 catch; [K,M]=LBtor6(p,R,rho); KM.K=K; KM.M=M; save(p.hofem.Kfn,'KM'); % assem
 end
else [K,M]=LBtor(p,R,rho); end % linear FEM
% assemble phi-different. matrix for phi-phase-cond, and transform to pBC
Dphi=convection(fem,gr,[1;0]); Dphi=[Dphi 0*Dphi; 0*Dphi Dphi];
p.mat.K=K; p.mat.M=[M 0*M; 0*M M]; p.mat.Dphi=filltrafo(p,Dphi);
[Dx,Dy]=fem.gradientMatrices(gr); E=center2PointMatrix(gr); Dx=E*Dx;
Dx=filltrafo(p,Dx); p.mat.Dx=[Dx 0*Dx; 0*Dx Dx];
```

```
function [K,M]=LBtor6(p,R,rho)  % LBO for torus based on 6-node triangles, hence
% assem6g instead assema, otherwise completely as LBtor.
po=getpte(p); th=po(2,:)'; n=p.np; [Kphi,M]=assem6g(p,[1 0; 0 0],1);
[Kth,~]=assem6g(p,[0*th' 0*th'; 0*th' R+rho*cos(th)'],1);
M=filltrafo(p,M);  dd=1./(R+rho*cos(th)); % build LBO and tranform to pBC:
K=filltrafo(p,spdiags(dd/rho^2,0,n,n)*Kth+spdiags(dd.^2,0,n,n)*Kphi);
```

Listing 4: `oosetfemops` and `LBtor6`.

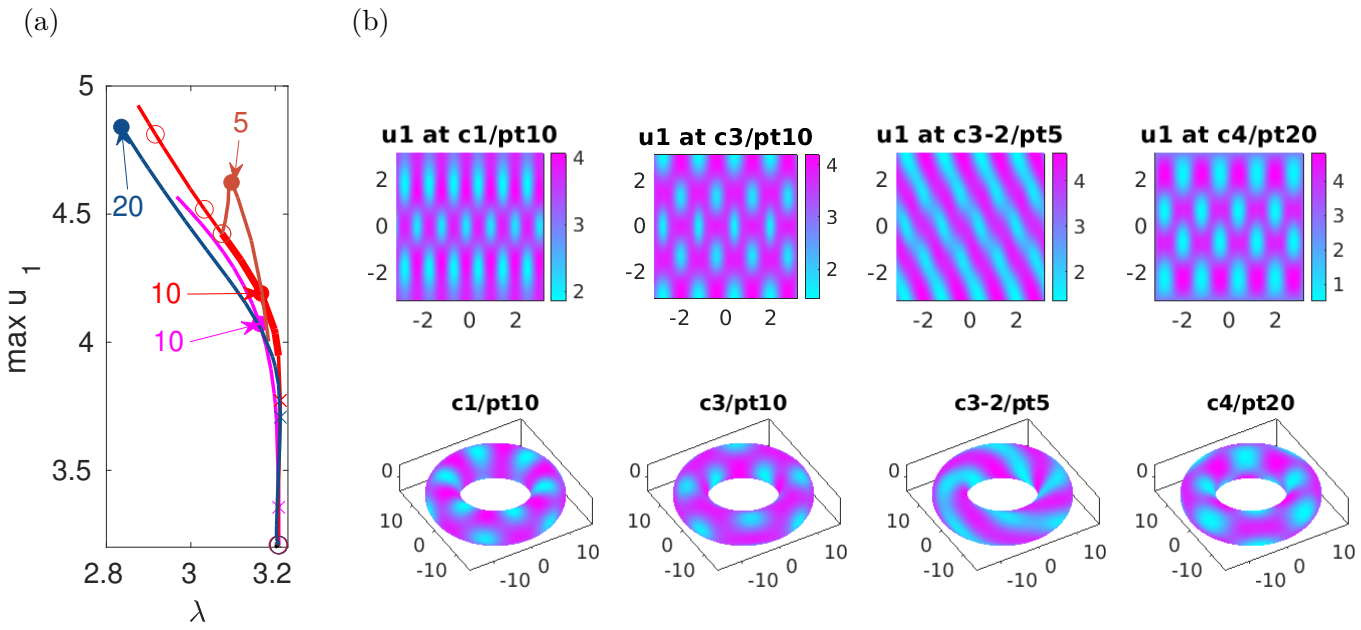

Figure 6: (15) on a torus with $(R,\rho)=(12,4)$, $\sigma=-0.1$, `np=4705` mesh points. Branches `c1` (magenta) `c3` (red, with stable patterns), `c3-2` (brown, secondary from 2nd BP of c3), and `c4` (blue), with sample plots.

Remark 4.1 a) The syntax for `c,a` in `[K,M]=assem6g(p,va)` is as in `fem.assema(grid,c,a,f)`, see [Uec21a, Remark A.1], where `va=varargin` can have the forms `[]` (empty), `va=c`, or `va=c,a`. The arguments `c,a` are then passed on to the element wise assembly `edmm6g`, and are treated as follows.

- If `va=[]`, then $c=1$ and $a=1$ and `assem6g` behaves like `assem6`.
- If `va=c`, then $a=1$ (assembly of standard M), and for `c` there are the following options:
  If `c` is a scalar, then $M^{-1}Ku$ corresponds to $c\Delta u$.
  If `c=[`$c_1,\ldots,c_{np}$`]` (one row=scalar function, given on the nodes), then $M^{-1}Ku$ corresponds to $\nabla\cdot(c\nabla u)$. The nodal values of c are interpolated to the element Gauss–points in `edmm6g`.
  If `c=[c1 c2; c3 c4]` (constant 2×2 matrix), then $M^{-1}Ku$ corresponds to $\nabla\cdot(\left(\begin{smallmatrix}c_1 & c_2\\ c_3 & c_4\end{smallmatrix}\right)\nabla u)$.
  If `c` is a $2\times 2n_p$ matrix of nodal values, then the first row of `c` contains c_1 and c_2, and the second row contains c_3 and c_4.
- If `va=c,a`, then a can be a number, or a row vector of nodal values.

b) All of the above applies analogously to `assem10g`.

c) Additionally, see the demos `hofem/acS` for (2) on a sphere parameterized by ϕ,ϑ–coordinates, where thus the LBO has a coordinate singularity at the poles, and `hofem/schnakcone` for (4) on a cone and the LBO is thus non–diagonal. See also [Uec21a, §10.1 and §10.2] for the respective results using the linear FEM. ⌋

A 1D

In 1D, we provide a *family* of higher order FEM, and we can more easily control the number of points during mesh adaptation. On the other hand, `pde2path` is rather intended as a 2D or 3D tool, and thus here we only briefly comment on the 1D case. In the demo `ac1D` we again consider (2), now over $\Omega=(-2\pi,2\pi)$ with DBCs.

To exploit the spectral FEM features of the `FSElib`, based on Lobatto-point discretizations of subintervals, we use the functions `p=two2lob(p)` and `[K,M]=assem1Dlob(p)`. In `two2lob` we use the grid points `x(1:np)=p.pdeo.grid.p(1:np)` as the end points of `ne=np-1` elements, and generate the new mesh `x(1:ng)` by introducing `npoly(j)-1` additional mesh points on the jth element. For sim-

plicity, in particular in mesh–adaptation, all entries in `npoly` are uniformly set to `p.hofem.femorder`, although non-uniform polynomial degrees can be used as well. `p=two2lob(p)` thus generates the data `p.hofem.xe` (end points of elements), `p.hofem.tri` (point indizes of elements), and updates `p.pdeo.grid` with the full mesh (e.g., for plotting). Based on this data, `[K,M]=assem1Dlob(p)` assembles the 1–component (Neumann–)Laplacian stiffness matrix K and the mass matrix M as usual (with $c = a = 1$). As before, these can be combined with other FEM operators (advection etc) based on the default `pde2path` FEM setup in `p.pdeo`.[2]

The script `ac1D/cmds` thus proceeds as usual, where additional to the domain size and initial discretization by n_x elements, we choose a FEM-order m. Calling `two2lob` in `acinit` then yields $mn_x + 1$ total mesh-points. We compute some primary BPs from the trivial branch, find that their accuracy quickly increases in m (and of course in n_x), and compute some primary bifurcating branches.

For adaptive mesh–refinement in the higher order setting, we locally overload the library function `oomeshada`, and there call `simplecoarselob` to first coarsen then refine a given mesh. In `simplecoarselob` we keep the first, the last, and every `p.nc.redf`–th point from the full mesh in `p.pdeo.grid.p`, and use the current solution on this coarse mesh for error estimation, here using the standard `pde2path` error estimator and element–to–refine selector `stane2rs` (controlled by `p.nc.sig`) to select elements to refine, which means introducing the elements midpoint. From the new set of elements we then again generate the full mesh via `p=two2lob(p)`. Using suitable `p.nc.redf` and `p.nc.sig`, for instance `p.nc.redf=p.hofem.femorder+1` and moderate `p.nc.sig` allows good control of the number of mesh–points under adaption, see the ends of `ac1D/cmds.m` and `ac1Dq/cmds.m`.

References

[FALD12] G. Formica, A. Arena, W. Lacarbonara, and H. Dankowicz. Coupling FEM with parameter continuation for analysis of bifurcations of periodic responses in nonlinear structures. *Journal of Computational and Nonlinear Dynamics*, 8(2), 2012.

[Lui11] S. H. Lui. *Numerical analysis of partial differential equations.* Pure and Applied Mathematics (Hoboken). John Wiley & Sons, Inc., Hoboken, NJ, 2011.

[Poz14] C. Pozrikidis. *Introduction to finite and spectral element methods using MATLAB®*. CRC Press, Boca Raton, FL, second edition, 2014.

[Prü21] U. Prüfert. OOPDE, `https://tu-freiberg.de/fakult1/nmo/pruefert`, 2021.

[RU19] J.D.M. Rademacher and H. Uecker. The OOPDE setting of pde2path – a tutorial via some Allen-Cahn models, 2019.

[SU17] G. Schneider and H. Uecker. *Nonlinear PDE – a dynamical systems approach*, volume 182 of *Graduate Studies Mathematics.* AMS, 2017.

[Uec19a] H. Uecker. Hopf bifurcation and time periodic orbits with pde2path – algorithms and applications. *Comm. in Comp. Phys*, 25(3):812–852, 2019.

[Uec19b] H. Uecker. Using `trullekrul` in `pde2path` – anisotropic mesh–adaptation for some Allen–Cahn models in 2D and 3D, Preprint, arXiv 1912.11130 , 2019.

[Uec21a] H. Uecker. *Numerical continuation and bifurcation in Nonlinear PDEs.* SIAM, 2021.

[Uec21b] H. Uecker. pde2path without finite elements, 2021. Tutorial on eqns on graphs, and spectral discretizations.

[2]For `femorder=2`, `two2lob` and `assem1Dlob` correspond to a quadratic FEM, with the element midpoints as Lobatto points. For this, there also is the function `[K,M,Kx]=assem1Dq(p)` which uses the quadratic FEM to also assemble `Kx` corresponding to ∂_x, see the demo `ac1Dq`.

[Uec21c] H. Uecker. `www.staff.uni-oldenburg.de/hannes.uecker/pde2path`, 2021.

[UWR14] H. Uecker, D. Wetzel, and J.D.M. Rademacher. pde2path – a Matlab package for continuation and bifurcation in 2D elliptic systems. *NMTMA*, 7:58–106, 2014.