The Equilibrium Point Toolbox

Harry Dankowicz
Department of Mechanical Science and Engineering
University of Illinois at Urbana-Champaign

Frank Schilder
Department of Mathematics
Technical University of Denmark

November 14, 2017

Contents

1 Introduction

The `'ep'` toolbox is a basic toolbox for continuation and bifurcation analysis of families of equilibria of evolution equations of the form

$$\dot{x} = F(x, p) \tag{1}$$

in terms of a vector of problem variables $x \in \mathbb{R}^n$, a vector of problem parameters $p \in \mathbb{R}^q$, and a nonlinear operator $F : \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^n$. For infinite-dimensional problems, the toolbox applies to suitable discretizations of x and F. The `'ep'` toolbox belongs to the `'ode'` toolbox family, and is modeled on the `'alg'` toolbox, described in *Recipes for Continuation*[1]. The `'ep'` toolbox supports detection of

- saddle-node bifurcations,
- Hopf bifurcations,
- neutral saddle points (optional and disabled by default), and
- branch and fold points (inherited from the associated atlas class),

as well as continuation along families of saddle-node and Hopf bifurcations. For continuation of equilibria, the `'ep'` toolbox supports the construction of the associated adjoint equations[2].

The toolbox user interface is defined by the `ep_read_solution` utility, which reads solution and toolbox data from disk, and by the toolbox constructors

- `ode_isol2ep` for continuation along a family of equilibria from an initial solution guess;
- `ode_ep2ep` for continuation along a family of equilibria from a saved solution point;
- `ode_BP2ep` for continuation along a family of equilibria from a branch point along a secondary branch;
- `ode_HB2HB` for continuation along a family of Hopf bifurcation points from a saved Hopf bifurcation point;
- `ode_SN2SN` for continuation along a family of saddle-node bifurcation points from a saved saddle-node bifurcation point.

The additional constructors `adjt_isol2ep`, `adjt_ep2ep`, and `adjt_BP2ep` contribute terms to the adjoint equations associated with the zero and monitor functions appended to a continuation problem by the `ode_isol2ep`, `ode_ep2ep`, and `ode_BP2ep` constructors, respectively.

[1] Dankowicz, H. & Schilder, F., *Recipes for Continuation*, Society for Industrial and Applied Mathematics, 2013.

[2] Li, M. & Dankowicz, H., *Staged Construction of Adjoints for Constrained Optimization of Integro-Differential Boundary-Value Problems*, in review, 2017.

Usage is illustrated in the following several examples. Each example corresponds to fully documented code in the `coco/ep/examples` folder in the COCO release. Slight differences between the code included below and the example implementations in `coco/ep/examples` show acceptable variations in the COCO syntax and demonstrate alternative solutions to construction and analysis. To gain further insight, please run the code to generate and explore figures and screen output.

Detailed information about COCO utilities deployed in these examples may be found in the document "Short Developer's Reference for COCO," available in the `coco/help` folder in the COCO release, and in *Recipes for Continuation*.

2 The cusp normal form – `cusp`

Consider the ordinary differential equation

$$\dot{x} = \kappa - x\left(\lambda - x^2\right) \tag{2}$$

in terms of the scalar problem variable $x \in \mathbb{R}$ and vector of problem parameters $p = (\kappa, \lambda) \in \mathbb{R}^2$. In this case, equilibrium solutions correspond to roots of the vector field

$$F(x, p) = \kappa - x\left(\lambda - x^2\right). \tag{3}$$

We proceed to encode the vector field and its Jacobians with respect to the problem variables and parameters in the anonymous functions `cusp`, `cusp_dx`, and `cusp_dp`, as shown in the following commands:

```
>> cusp    = @(x,p) p(1)-x*(p(2)-x^2);
>> cusp_dx = @(x,p) 3*x^2-p(2);
>> cusp_dp = @(x,p) [1 -x];
```

We compute a family of equilibria under variations in κ by invoking the `coco` entry-point function as shown in the sequence of commands below:

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'vectorized', false);
>> ode_fcns = {cusp, cusp_dx, cusp_dp};
>> prob = ode_isol2ep(prob, '', ode_fcns{:}, 0, {'ka' 'la'}, [0; 0.5]);
>> coco(prob, 'cusp1', [], 1, {'ka' 'la'}, [-0.5 0.5]);
```

Here, the `coco_prob` core utility assigns an empty continuation problem structure to `prob`. The `coco_set` core utility assigns the non-default value of `false` to the `'vectorized'` setting of the `'ode'` toolbox family, in order to indicate the non-vectorized encoding of the vector field and its Jacobians. The content of `prob` is then modified by the `ode_isol2ep` toolbox constructor. This stores a representation of the continuation problem with initial solution guess $(x, \kappa, \lambda) = (0, 0, 0.5)$, including the definition of two inactive continuation parameters, denoted by `'ka'` and `'la'`, that track/constrain the values of κ and λ, respectively.

The call to the `coco` entry-point function identifies the run by the string identifier `'cusp1'`, recognizes by the empty bracket the encoding of the corresponding extended continuation problem and the initial assignment of inactive continuation parameters in `prob`,

and identifies the desired dimension of the solution manifold by the integer 1. As the dimensional deficit[3] of the restricted continuation problem encoded by the call to `ode_isol2ep` equals 0, it is necessary to release[4] one of the inactive continuation parameters, in order to obtain a restricted continuation problem with dimensional deficit of 1. The subsequent reference to `'ka'` and `'la'`, in that order, implies that the continuation parameter `'ka'` is active and varying throughout the run, while `'la'` remains inactive and constant. Values of both parameters are printed to screen during continuation. The final argument defines bounds on the computational domain, restricting continuation to the range $-0.5 \leq \kappa \leq 0.5$. We visualize the result of continuation using the core bifurcation data visualizer `coco_plot_bd` as shown here:

```
>> figure(1); clf
>> thm = struct('special', {{'SN'}});
>> coco_plot_bd(thm, 'cusp1', 'ka', 'x')
>> grid on
```

This produces a graph of values of x versus `'ka'` with default formating of branches of stable and unstable equilbria, respectively. The information in the `thm` theme structure overrides defaults encoded in the toolbox utility `ep_plot_theme` in order to include a marker at the saddle-node bifurcation along the solution manifold.

The two saddle-node bifurcations detected and located during continuation may serve as starting points for continuation along families of saddle-node bifurcation points. To this end, we use the `coco_bd_read` core utility to extract information from the bifurcation data file stored during continuation, as shown below. The subsequent call to the `coco_bd_labs` utility returns an array of integers associated with the solution files corresponding to the saddle-node bifurcation points.

```
>> bd   = coco_bd_read('cusp1');
>> labs = coco_bd_labs(bd, 'SN');
```

We use the solution chart structure and data array stored in the first of these solution files to construct a corresponding continuation problem structure.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'vectorized', false);
>> prob = ode_SN2SN(prob, '', 'cusp1', labs(1));
>> coco(prob, 'cusp2', [], 1, {'ka' 'la'}, [-0.5 0.5]);
```

In this case, the dimensional deficit of the restricted continuation problem encoded by the

[3]Dimensional deficit: difference between the number of continuation variables and active continuation parameters (here equal to the number of problem variables plus the number of problem parameters, i.e., $1 + 2 = 3$), and the number of imposed equations (here equal to the zero problem and the assignment of the values of the problem parameters to the two inactive continuation parameters, i.e., $1 + 2 = 3$).

[4]To release: to include in the list of continuation parameters passed to COCO in the next-to-last argument. Inactive continuation parameters are activated (i.e., allowed to vary during continuation), to the extent required by the manifold dimension, in the order listed. Surplus inactive continuation parameters remain inactive.

ode_SN2SN constructor[5] equals -1. A one-dimensional solution manifold results by releasing two inactive continuation parameters. It follows that, in this run, both 'ka' and 'la' are active and varying. The following call to coco_plot_bd visualizes the result of continuation.

```
>> figure(2); clf
>> coco_plot_bd('cusp2')
>> grid on
```

In this case, the defaults encoded in ep_plot_theme ensure that, by omission of additional arguments to coco_plot_bd, the horizontal axis represents the first output parameter 'ka', and the vertical axis represents second output parameter 'la'.

Exercises

1. Verify that the sequence of commands

   ```
   >> ode_fcns = {cusp, cusp_dx, cusp_dp};
   >> prob = ode_isol2ep(prob, '', ode_fcns{:}, 0, {'ka' 'la'}, [0; 0.5]);
   >> coco(prob, 'cusp1', [], 1, {'ka' 'la'}, [-0.5 0.5]);
   ```

   may be replaced by either of the following single calls to the coco entry-point function:

   ```
   >> coco(prob, 'cusp1', @ode_isol2ep, cusp, cusp_dx, cusp_dp, 0,  ...
        {'ka' 'la'}, [0; 0.5], 1, {'ka' 'la'}, [-0.5 0.5]);
   >> coco(prob, 'cusp1', 'ode', 'isol', 'ep', cusp, cusp_dx, cusp_dp, 0, ...
        {'ka' 'la'}, [0; 0.5], 1, {'ka' 'la'}, [-0.5 0.5]);
   ```

   In the second of these alternative calling syntaxes, the string 'ode' identifies the toolbox family, the string 'isol' identifies the starting point of continuation as one obtained from an initial solution guess, and the string 'ep' identifies the type of solutions computed by continuation.

2. Verify that the call to the coco_bd_read core utility may be omitted by assigning the output of the coco entry-point function directly to the corresponding variable.

3. Verify that the inclusion of function handles to the Jacobians with respect to the problem variables and problem parameters is optional.

4. Use ode_SN2SN to perform continuation over the computational domain $-1 \leq \kappa \leq -0.5$ along a family of saddle-node bifurcations starting from the second solution stored to disk in the 'cusp2' run, and visualize the result using the coco_plot_bd bifurcation data visualizer.

---

[5]The condition for saddle-node bifurcations implemented in the current version of the 'ep' toolbox adds $2n$ additional continuation variables and imposes $2n+1$ additional equations. Hence, the dimensional deficit is reduced by 1 relative to that of the equilibrium point restricted continuation problem.

3 Bratu's problem – `bratu`

Consider the scalar boundary-value problem

$$u_t = u_{\xi\xi} + \lambda u + \mu e^u,\ u(0,t) = u(1,t) = 0 \tag{4}$$

on the two-dimensional domain $(t, \xi) \in \mathbb{R}\times[0,1]$, in terms of the vector of problem parameters $p = (\mu, \lambda) \in \mathbb{R}^2$. We arrive at a finite-dimensional, ordinary differential equation of the form

$$\dot{x} = F(x, p) \tag{5}$$

by discretizing the unknown function $u(\xi, t)$ in terms of its nodal values $u_i(t)$ at $\xi = \xi_i := i/N$, for $i = 1, \ldots, N-1$, and the one-dimensional spatial Laplace operator by a mid-point finite difference formula

$$u_{\xi\xi}(\xi_i, t) \mapsto N^2\big(u_{i-1}(t) - 2u_i(t) + u_{i+1}(t)\big), \tag{6}$$

where $u_0(t) = u_N(t) \equiv 0$. Specifically,

$$\dot{x} = N^2 \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix} \cdot x + \lambda x + \mu \begin{pmatrix} e^{u_1} \\ \vdots \\ e^{u_{N-1}} \end{pmatrix} \tag{7}$$

in terms of the vector of problem variables

$$x = \begin{pmatrix} u_1 \\ \vdots \\ u_{N-1} \end{pmatrix}. \tag{8}$$

We proceed to encode the vector field on the right-hand side of (7) and its Jacobians with respect to the problem variables and parameters in the anonymous functions `bratu`, `bratu_dx`, and `bratu_dp`, as shown in the following commands:

```
>> N = 20;
>> D = diag(-2*ones(N-1,1)) + diag(ones(N-2,1),-1) + diag(ones(N-2,1),1);
>> D = N^2*D;
>> bratu    = @(u,p) D*u + p(2)*u            + p(1)*exp(u);
>> bratu_dx = @(u,p) D   + p(2)*eye(N-1,N-1) + p(1)*diag(exp(u));
>> bratu_dp = @(u,p) [exp(u) u];
```

We compute a family of equilibria under variations in μ by invoking the `coco` entry-point function as shown in the sequence of commands below:

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'vectorized', false);
>> prob = coco_set(prob, 'cont', 'PtMX', 50);
>> ode_fcns = {bratu, bratu_dx, bratu_dp};
>> ode_args = {ode_fcns{:}, zeros(N-1,1), {'mu' 'la'}, [0; 0]};
>> cont_args = {1, {'mu' 'ep.test.SN'}, [0 4]};
>> bd1 = coco(prob, 'bratu1', @ode_isol2ep, ode_args{:}, cont_args{:});
```

Here, the coco_prob core utility assigns an empty continuation problem structure to prob. The coco_set core utility assigns the value of false to the 'vectorized' setting of the 'ode' toolbox family, in order to indicate the non-vectorized encoding of the vector field and its Jacobians. We encode the value of 50 for the 'PtMX' setting of the 'cont' toolbox, imposing an upper bound of 50 continuation steps in each direction along the solution manifold. The call to the coco entry-point function identifies the run by the string identifier 'bratu1', and uses the ode_isol2ep toolbox constructor to temporarily modify the content of prob to include a representation of the continuation problem with initial solution guess $(u_1, \ldots, u_{N-1}, \mu, \lambda) = (0, \ldots, 0, 0, 0)$, as well as two inactive continuation parameters, denoted by 'mu' and 'la', that track/constrain the values of μ and λ, respectively.

In the cont_args variable, the integer 1 identifies the desired dimension of the solution manifold. As the dimensional deficit of the restricted continuation problem encoded by the call to ode_isol2ep equals 0, it is necessary to release one of the inactive continuation parameters, in order to obtain a restricted continuation problem with dimensional deficit of 1. The subsequent reference to 'mu' and 'ep.test.SN' implies that the continuation parameter 'mu' is active and varying throughout the run. The inclusion of the 'ep.test.SN' nonembedded continuation parameter ensures that the value of the saddle-node monitor function is printed to screen (it is included with the bifurcation data stored during continuation by default). Finally, the argument [0 4] defines bounds on the computational domain, restricting continuation to the range $0 \leq \mu \leq 4$.

We may restart continuation from the final point on the solution manifold found in the previous run using the ode_ep2ep toolbox constructor, as shown in the following sequence of commands.

```
>> labs = coco_bd_labs(bd1, 'EP');
>> ode_args = {'bratu1', labs(end)};
>> cont_args = {1, {'mu' 'ep.test.SN'}, [0 4]};
>> bd2 = coco(prob, 'bratu2', @ode_ep2ep, ode_args{:}, cont_args{:});
```

We visualize the result of continuation by using the coco_plot_bd bifurcation data visualizer, as shown below.

```
>> figure(1); clf
>> thm = struct('special', {{'SN'}});
>> coco_plot_bd(thm, 'bratu2')
>> grid on
```

The defaults encoded in ep_plot_theme ensure that, by omission of additional arguments to coco_plot_bd, the horizontal axis represents the first output parameter, i.e., 'mu', and the vertical axis represents the Euclidean norm $\sqrt{u_1^2 + \cdots + u_{N-1}^2}$.

The saddle-node bifurcation detected and located during continuation may serve as a starting point for continuation along a family of saddle-node bifurcation points, as shown below.

```
>> labs = coco_bd_labs(bd1, 'SN');
>> prob = coco_set(prob, 'cont', 'PtMX', 100);
>> ode_args = {'bratu1', labs(1)};
```

```
>> cont_args = {1, {'mu' 'la'}, {[-4 4] [-2 20]}};
>> bd3 = coco(prob, 'bratu3', @ode_SN2SN, ode_args{:}, cont_args{:});
```

In this case, the restricted continuation problem encoded by the call to the `ode_SN2SN` constructor has dimensional deficit equal to -1. A one-dimensional solution manifold results by releasing two inactive continuation parameters. It follows that, in this run, both `'mu'` and `'la'` are active and varying. Notably, the branch point detected at $\mu \approx 0$ corresponds to the intersection with a branch of approximate eigenfunctions of the Laplace operator with the given boundary conditions corresponding to the approximate eigenvalue $\lambda \approx 9.8493$.

The following commands generate three-dimensional representations of the original branch of equilibrium solutions and the saddle-node bifurcation curve.

```
>> figure(2); clf; hold on
>> coco_plot_bd('bratu1', 'la', 'mu', '||x||_2')
>> thm = struct('special', {{'BP'}});
>> coco_plot_bd(thm, 'bratu3', 'la', 'mu', '||x||_2')
>> hold off; grid on; view(3)
```

Exercises

1. Use the `ode_ep2ep` constructor to restart continuation along branches of equilibria from each of the solution points on the saddle-node bifurcation curve stored in the last run, and overlay this family of solution curves on the visualization of the backbone saddle-node curve.

2. Experiment with different values of N and explore spurious results for small values of N, as well as convergence properties as $N \to \infty$. How does the value of λ at the branch point with $\mu \approx 0$ depend on N? At what rate does it converge to the theoretical value of π^2, if at all?

3. Use the `ode_BP2ep` constructor to restart continuation from the branch point found in the last run along the secondary branch of equilibrium solutions, corresponding to approximate eigenfunctions of the Laplace operator with zero Dirichlet boundary conditions.

4. What effect does the following change of the order of continuation parameters have on the computation of `bd1` in the example on page 7?

   ```
   cont_args = {1, {'ep.test.SN' 'mu'}, {[], [0 4]}};
   ```

---

4 A Brusselator model – **`brus`**

Consider the boundary-value problem given by the coupled differential equations

$$u_t = \delta u_{\xi\xi} + \alpha + u^2 v - (\beta + 1)u,\ v_t = \rho\delta v_{\xi\xi} + \beta u - u^2 v \tag{9}$$

and boundary conditions

$$u(0,t) = u(1,t) = \alpha,\ v(0,t) = v(1,t) = \beta/\alpha \tag{10}$$

on the two-dimensional domain $(t,\xi) \in \mathbb{R}\times[0,1]$, in terms of the vector of problem parameters $p = (\alpha,\beta,\delta,\rho) \in \mathbb{R}^4$. We arrive at a finite-dimensional, differential-algebraic problem of the form

$$\dot{x}_1 = F_1(x,p),\ 0 = F_2(x,p),\ x = (x_1,x_2) \tag{11}$$

by discretizing the unknown functions $u(\xi,t)$ and $v(\xi,t)$ in terms of their nodal values $u_i(t)$ and $v_i(t)$ at $\xi = \xi_i := i/N$, for $i = 0,\ldots,N$, and the one-dimensional Laplace operator for $i = 1,\ldots,N-1$ by a mid-point finite difference formula:

$$u_{\xi\xi}(\xi_i,t) \mapsto N^2\big(u_{i-1}(t) - 2u_i(t) + u_{i+1}(t)\big), \tag{12}$$

$$v_{\xi\xi}(\xi_i,t) \mapsto N^2\big(v_{i-1}(t) - 2v_i(t) + v_{i+1}(t)\big). \tag{13}$$

We proceed to encode the vector field (F_1,F_2) and its Jacobians with respect to the problem variables and problem parameters in the anonymous functions `bruss`, `bruss_dx`, and `bruss_dp`, respectively, as shown in the following commands:

```
>> N  = 20;
>> X  = 1:N+1;
>> Y  = N+1+X;
>> B  = [1; zeros(N-1,1); 1];
>> BX = repmat(B,1,N+1);
>> BP = repmat(B,1,4);
>> C  = [0; ones(N-1,1); 0];
>> CX = repmat(C,1,N+1);
>> CP = repmat(C,1,4);
>> D  = diag([0 -2*ones(1,N-1) 0]) + ...
     diag([ones(1,N-1) 0],-1) + ...
     diag([0 ones(1,N-1)],1);
>> D  = N^2*D;
>> ID = eye(N+1,N+1);
>> O  = ones(N+1,1);
>> ZE = zeros(N+1,1);
>> bruss  = @(u,p) [
     C.*(p(3)*D*u(X)+p(1)+u(X).^2.*u(Y)-(p(2)+1)*u(X))+B.*(p(1)-u(X))
     C.*((p(3)*p(4))*D*u(Y)+p(2)*u(X)-u(X).^2.*u(Y))+B.*(p(2)/p(1)-u(Y))
   ];
>> bruss_dx = @(u,p) [
     CX.*(p(3)*D+2*diag(u(X).*u(Y))-(p(2)+1)*ID)-BX.*ID ...
     CX.*(diag(u(X).^2))
     CX.*(p(2)*ID-2*diag(u(X).*u(Y))) ...
     CX.*((p(3)*p(4))*D-diag(u(X).^2))-BX.*ID
   ];
>> bruss_dp = @(u,p) [
     C+B  C.*(-u(X)) ...
     C.*(D*u(X))  ZE
     -B.*(p(2)/p(1)^2)  C.*(u(X))+B.*(1/p(1)) ...
     C.*(p(4)*D*u(Y))  C.*(p(3)*D*u(Y))
   ];
```

We compute a family of equilibrium solutions under variations in β by invoking the `coco` entry-point function as shown in the sequence of commands below.

```
>> p0 = [1; 3; 0.075; 1];
>> u0 = [p0(1)*ones(N+1,1); (p0(2)/p0(1))*ones(N+1,1)];
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'vectorized', false);
>> ode_fcns = {bruss, bruss_dx, bruss_dp};
>> ode_args = {ode_fcns{:}, u0, {'al' 'be' 'de' 'ro'}, p0};
>> cont_pars = {'be' 'ep.test.SN' 'ep.test.HB' 'ep.test.USTAB' ...
     'atlas.test.FP'};
>> cont_args = {1, cont_pars, [2 7]};
>> bd1 = coco(prob, 'brus1', @ode_isol2ep, ode_args{:}, cont_args{:});
```

Here, the initial solution guess contained in `p0` and `u0` satisfies the boundary conditions, but not the governing differential equations. The inclusion of the `'ep.test.SN'`, `'ep.test.HB'`, `'ep.test.USTAB'`, and `'atlas.test.FP'` nonembedded continuation parameters ensures that the values of the saddle-node, Hopf, stability indicator, and fold-point monitor functions are printed to screen (they are included with the bifurcation data stored during continuation by default).

We can restart continuation from each of the two branch points found in the previous run using the `ode_BP2ep` toolbox constructor, as shown in the following sequence of commands.

```
>> labs = coco_bd_labs(bd1, 'BP');
>> for lab = labs
     ode_args = {'brus1', lab};
     cont_pars = {'be' 'ep.test.SN' 'ep.test.HB' 'ep.test.USTAB' ...
       'atlas.test.FP'};
     cont_args = {1,  cont_pars, [2 7]};
     run = sprintf('brus2_%02d', lab);
     coco(prob, run, @ode_BP2ep, ode_args{:}, cont_args{:});
   end
```

We visualize the result of the various continuation runs with the `coco_plot_bd` bifurcation data visualizer.

```
>> figure(1); clf; hold on
>> thm = struct('special', {{'BP', 'HB'}});
>> coco_plot_bd(thm, 'brus1')
>> thm = struct('special', {{'FP', 'HB'}});
>> for lab=labs
     coco_plot_bd(thm, sprintf('brus2_%02d', lab))
>> end
>> hold off; grid on
```

As before, by omission of the second and third arguments, the horizontal axis defaults to the primary output parameter `'be'` and the vertical axis defaults to the Euclidean norm $\sqrt{u_0^2 + \cdots + u_N^2 + v_0^2 + \cdots + v_N^2}$.

The Hopf and saddle-node bifurcation points detected and located during the last set of continuation runs may serve as starting points for continuation along families of Hopf and saddle-node bifurcation points. In the calls below, the dimensional deficits of the restricted

continuation problems encoded by either of the `ode_SN2SN` and `ode_HB2HB` constructors equal[6] -1. In either case, a one-dimensional solution manifold results by releasing two inactive continuation parameters. It follows that, in each of the runs below, the continuation parameters `'be'` and `'de'` are active and varying.

```
>> HBlabs = coco_bd_labs(bd1, 'HB');
>> ode_args = {'brus1', HBlabs(1)};
>> cont_args = {1, {'de' 'be' 'ep.test.BT'}, {[0 0.2] [2 7]}};
>> coco(prob, 'brus_HB1', @ode_HB2HB, ode_args{:}, cont_args{:});
>> rrun   = sprintf('brus2_%02d', labs(1));
>> bd     = coco_bd_read(rrun);
>> HBlabs = coco_bd_labs(bd, 'HB');
>> vals   = coco_bd_vals(bd, HBlabs, 'be');
>> [v i]  = min(vals);
>> ode_args  = {rrun, HBlabs(i)};
>> cont_args = {1, {'de' 'be'}, {[0 0.2] [2 7]}};
>> coco(prob, 'brus_HB2', @ode_HB2HB, ode_args{:}, cont_args{:});
>> SNlabs = coco_bd_labs(bd, 'SN');
>> vals   = coco_bd_vals(bd, SNlabs, 'be');
>> [v i]  = min(vals);
>> prob = coco_set(prob, 'cont', 'PtMX', 200);
>> ode_args  = {rrun, SNlabs(i)};
>> cont_args = {1, {'de' 'be'}, {[0 0.2] [2 7]}};
>> coco(prob, 'brus_SN', @ode_SN2SN, ode_args{:}, cont_args{:});
```

We note the reference to the `'ep.test.BT'` nonembedded continuation parameter in the call to the `coco` entry-point function, in order to ensure that the value of the Bogdanov-Takens bifurcation test function is printed to screen during continuation along the Hopf bifurcation curve. The code also illustrates the use of the `coco_bd_vals` utility for extracting the values of the continuation parameter `'be'` at the labeled solution points. We visualize the results of continuation using the sequence of commands shown below.

```
>> figure(2); clf; hold on
>> coco_plot_bd('brus_SN', 'be', 'de', '||x||_2')
>> thm = struct('special', {{'BTP'}});
>> coco_plot_bd(thm, 'brus_HB1', 'be', 'de', '||x||_2')
>> coco_plot_bd(thm, 'brus_HB2', 'be', 'de', '||x||_2')
>> hold off; grid on; view(3)
```

where `'||x||_2'` denotes the Euclidean norm $\sqrt{u_0^2 + \cdots + u_N^2 + v_0^2 + \cdots + v_N^2}$.

Exercises

1. Verify the encoding of the vector fields F_1 and F_2 and their Jacobians obtained by discretization of the spatial boundary-value problem governing equilibrium solutions

[6] The condition for Hopf bifurcation equilibrium points implemented in the current version of the `'ep'` toolbox adds $3n + 1$ additional continuation variables and imposes $3n + 2$ additional equations. Hence, the dimensional deficit is reduced by 1 relative to that of the equilibrium point restricted continuation problem.

of the coupled Brusselator equations.

2. Experiment with different values of N and explore spurious results obtained during continuation along a family of equilibria for small values of N, as well as convergence properties as $N \to \infty$.

3. Consider the encoding of the Brusselator boundary conditions in the vector field F_2. Comment on the relationship between solutions to the differential-algebraic equations

$$\dot{x}_1 = F_1(x, p),\ 0 = F_2(x, p), \tag{14}$$

and solutions of the ordinary differential equations

$$\dot{x}_1 = F_1(x, p),\ \dot{x}_2 = F_2(x, p), \tag{15}$$

for arbitrary initial conditions.

4. Perform forward integration with one of MATLAB's ODE integrators using the vector field encoded in `bruss` and comment on the way in which the algebraic conditions are enforced during integration. Visualize the solutions in time and consider parameter values on either side of a Hopf bifurcation point. Try several initial conditions and discuss uniqueness, stability of solutions, and smoothness of solution profile.

---

5 The Laplace Operator – `pdeeig`

Consider the boundary-value problem

$$\triangle u = -\lambda(u + \mu e^u), u_{\mathbb{R}^2 \setminus D} = 0 \tag{16}$$

in terms of the vector of problem parameters $p = (\mu, \lambda) \in \mathbb{R}^2$. Here, the set D is the L-shaped portion of the open rectangle $R := \{(x, y) \,|\, 0 < x < 2, 0 < y < 3\}$ obtained by removing the rectangle $\{(x, y) \,|\, 0 < x \leq 1, 0 < y \leq 2\}$. Let the unknown function $u(x, y)$ be represented by its nodal values $u_{i,j}$ on the grid (x_i, y_j), $i = 1, \ldots, 2N + 1$, $j = 1, \ldots, 3N + 1$, where $x_i = (i - 1)/N$ and $y_j = (j - 1)/N$. Moreover, approximate the Laplacian on the set D by the five-point finite-difference approximation

$$\triangle u(x_i, y_j) \mapsto N^2 \left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}\right). \tag{17}$$

We proceed to encode the governing algebraic equations and their Jacobians with respect to the unknown nodal values and the problem parameters in the anonymous functions `pdeeig`, `pdeeig_dx`, and `pdeeig_dp`, as shown in the following commands:

```
>> N = 20;
>> P = 3*N+1;
>> Q = 2*N+1;
>> X = reshape(1:P*Q, P, Q);
>> Mask            = true(P,Q);
>> Mask(1:end,1)   = false;
>> Mask(1:end,end) = false;
>> Mask(1,1:end)   = false;
>> Mask(end,1:end) = false;
>> Mask(1:2*N+1,1:N+1) = false;
>> rows = X(Mask);
>> cols = rows;
>> o    = ones(numel(rows),1);
>> C    = sparse(rows, cols, o, P*Q, P*Q);
>> D    = sparse(rows, cols, -4*o, P*Q, P*Q);
>> cols = X(circshift(Mask, [0 1]));
>> D    = D + sparse(rows, cols, o, P*Q, P*Q);
>> cols = X(circshift(Mask, [0 -1]));
>> D    = D + sparse(rows, cols, o, P*Q, P*Q);
>> cols = X(circshift(Mask, [1 0]));
>> D    = D + sparse(rows, cols, o, P*Q, P*Q);
>> cols = X(circshift(Mask, [-1 0]));
>> D    = D + sparse(rows, cols, o, P*Q, P*Q);
>> D    = N^2*D;
>> rows = X(~Mask);
>> cols = rows;
>> o    = ones(numel(rows),1);
>> B    = sparse(rows, cols, o, P*Q, P*Q);
>> Id   = speye(P*Q, P*Q);
>> pdeeig    = @(u,p) D*u + p(2)*(C*u + p(1)*(C*exp(u))) - B*u;
>> pdeeig_dx = @(u,p) D + p(2)*(C + p(1)*(C*spdiags(exp(u),0,Id))) - B;
>> pdeeig_dp = @(u,p) [p(2)*(C*exp(u)) C*u+p(1)*(C*exp(u))];
```

For $\mu = 0$, a trivial solution family is obtained with $u_{i,j} = 0$, $\forall i, j$ as shown below.

```
>> p0 = [0; 0.1];
>> u0 = zeros(P*Q,1);
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'vectorized', false);
>> prob = coco_set(prob, 'ep', 'bifus', false);
>> prob = coco_set(prob, 'cont', 'PtMX', 200);
>> ode_args = {pdeeig, pdeeig_dx, pdeeig_dp, u0, {'mu' 'la'}, p0};
>> cont_args = {1, 'la', [-5 50]};
>> bd = coco(prob, 'pdeeig1', @ode_isol2ep, ode_args{:}, cont_args{:});
```

We turn off bifurcation detection in order to reduce the demand on run-time memory. Branch points found during continuation along this family correspond to approximate eigenvalues of the Laplacian on the domain D with zero Dirichlet boundary conditions. We may restart continuation from each of these branch points in order to generate the corresponding approximate eigenfunctions, according to some normalization scheme.

```
>> labs = coco_bd_labs(bd, 'BP');
>> prob = coco_set(prob, 'cont', 'PtMX', [10 0]);
>> prob = coco_set(prob, 'cont', 'h0', 1);
```

```
>> for lab = labs
     prob2 = ode_BP2ep(prob, '', 'pdeeig1', lab);
     [fdata uidx] = coco_get_func_data(prob2, 'ep', 'data', 'uidx');
     xidx         = uidx(fdata.ep_eqn.x_idx);
     prob2        = coco_add_func(prob2, 'norm_x', @norm_x, [], ...
       'regular', 'norm_x', 'uidx', xidx);
     prob2 = coco_add_event(prob2, 'UZ', 'BP', 'norm_x', 1);
     runid = sprintf('pdeeig2_%02d', lab);
     bd2   = coco(prob2, runid, [], 1, {'la' 'norm_x'});
     figure(1); clf
     coco_plot_sol(struct('plot_sol', @ef_plot, 'N', N), runid, '')
     view([-125 60])
     drawnow
   end
```

Here, the `coco_get_func_data` utility is used to extract the integer indices associated with the nodal values in the array of continuation variables. Together with the content of the `ep_eqn` field of the `'ep'` toolbox data structure (see the documentation for the `ep_add` toolbox interface function for more details), these are used to identify the function dependency index set[7] for the nonembedded monitor function `norm_x` encoded in the program file `norm_x.m`, shown below.

```
function [data y] = norm_x(opts, data, u)
  y = norm(u);
end
```

A boundary event is then detected when the corresponding continuation parameter crosses 1.

The call to the `coco_plot_sol` core utility includes a plotting theme structure that specifies the problem-specific graphing action to be applied to the solution associated with the `'UZ'` event in the run `runid`. The encoding of `ef_plot` below uses the value of N to generate a surface mesh representation of the eigenfunction.

```
function thm = ef_plot(thm, ~, ~, bd, ~)

persistent X Y P Q N

if isempty(X) || ~(N==thm.N)
  N = thm.N;
  P = 3*N+1;
  Q = 2*N+1;
  [X, Y] = meshgrid(linspace(0,2,Q), linspace(0,3,P));
end
lab = coco_bd_labs(bd, 'UZ');
U   = coco_bd_val(bd, lab, 'x');
U   = reshape(U, P, Q);
mesh(X,Y,U)

thm.xlab = 'x'; thm.ylab = 'y'; thm.zlab = 'u';

end
```

[7]Function dependency index set: an ordered set of integer indices associated with elements of the vector of continuation variables that constitute the arguments of a COCO-compatible function encoding.

Exercises

1. Verify the encoding of the vector field and its Jacobians obtained by discretization of the spatial boundary-value problem governing eigenfunctions of the Laplace operator.

2. Comment on the reason for defining the discretization on the entire rectangle R instead of only on the L-shaped subset D. How exactly is this implemented?

3. Use the approach in this section to compute the eigenfunctions and eigenvalues for the Laplace operator on a circular domain. Use conformal mapping techniques to extend the approach to other nontrivial domains.

4. Comment on the use of the Euclidean norm of the vector of problem variables for the normalization of the eigenfunctions. What happens when you vary N? Implement a modification to the function `norm_x` that reduces to the L_2-norm for $N \to \infty$ and discuss your observations.

---

6 Chemical oscillations – `chemosc`

Let $z = 1 - x_1 - x_2 - x_3$ and consider the quadratic vector field

$$F(x,p) = \begin{pmatrix} 2p_1 z^2 - 2p_5 x_1^2 - p_3 x_1 x_2 \\ p_2 z - p_6 x_2 - p_3 x_1 x_2 \\ p_4 z - p_4 p_7 x_3 \end{pmatrix} \tag{18}$$

corresponding to the Bykov-Yablonskii-Kim model[8] of oxidation of carbon monoxide on platinum, expressed in terms of the vector of problem variables $x \in \mathbb{R}^3$ and vector of problem parameters $p \in \mathbb{R}^7$. We proceed to encode vectorized implementations of the vector field and its Jacobians with respect to the problem variables and parameters in the functions `bykov`, `bykov_dx`, and `bykov_dp` shown below.

```
function f = bykov(x, p)

p1 = p(1,:);
p2 = p(2,:);
p3 = p(3,:);
p4 = p(4,:);
p5 = p(5,:);
p6 = p(6,:);
p7 = p(7,:);
```

[8] See "Tutorial IV: Two-parameter bifurcation analysis of equilibria and limit cycles with MATCONT," by Yu.A. Kuznetsov, from September 20, 2011, available at `http://www.staff.science.uu.nl/~kouzn101/NBA/LAB4.pdf`.

```
x1 = x(1,:);
x2 = x(2,:);
x3 = x(3,:);

z = 1 - x1 - x2 - x3;
f = [2*p1.*z.^2 - 2*p5.*x1.^2 - p3.*x1.*x2;
     p2.*z - p6.*x2 - p3.*x1.*x2;
     p4.*z - p7.*p4.*x3];

end


function J = bykov_dx(x, p)

p1 = p(1,:);
p2 = p(2,:);
p3 = p(3,:);
p4 = p(4,:);
p5 = p(5,:);
p6 = p(6,:);
p7 = p(7,:);

x1 = x(1,:);
x2 = x(2,:);
x3 = x(3,:);

z = 1 - x1 - x2 - x3;
J = zeros(3,3,numel(z));

J(1,1,:) = -4*p5.*x1 - p3.*x2 - 4*p1.*z;
J(1,2,:) = -p3.*x1 - 4*p1.*z;
J(1,3,:) = -4*p1.*z;
J(2,1,:) = -p2 - p3.*x2;
J(2,2,:) = -p2 - p6 - p3.*x1;
J(2,3,:) = -p2;
J(3,1,:) = -p4;
J(3,2,:) = -p4;
J(3,3,:) = -p4 - p4.*p7;

end


function J = bykov_dp(x, p)

p4 = p(4,:);
p7 = p(7,:);

x1 = x(1,:);
x2 = x(2,:);
x3 = x(3,:);

z = 1 - x1 - x2 - x3;
J = zeros(3,7,numel(z));

J(1,1,:) = 2*z.^2;
```

```
J(1,3,:) = -x1.*x2;
J(1,5,:) = -2*x1.^2;
J(2,2,:) = z;
J(2,3,:) = -x1.*x2;
J(2,6,:) = -x2;
J(3,4,:) = z - p7.*x3;
J(3,7,:) = -p4.*x3;

end
```

We compute families of equilibria under variations in p_2 for $p_7 = 0.4$, $p_7 = 0.15$, and $p_7 = 2.0$, respectively, and fixed values of the other problem parameters, by invoking the `coco` entry-point function as shown in the sequence of commands below.

```
>> x0     = [0.001137; 0.891483; 0.062345];
>> pnames = {'p1' 'p2' 'p3' 'p4' 'p5' 'p6' 'p7'};
>> p0     = [2.5; 2.204678; 10; 0.0675; 1; 0.1; 0.4];
>> prob = coco_prob();
>> prob = coco_set(prob, 'ep', 'NSA', true);
>> ode_fcns  = {@bykov, @bykov_dx, @bykov_dp};
>> ode_args  = {ode_fcns{:}, x0, pnames};
>> cont_args = {1, 'p2', [0.4 3]};
>> coco(prob, 'p7=0.4', @ode_isol2ep, ode_args{:}, p0, cont_args{:});
>> p0(7) = 0.15;
>> coco(prob, 'p7=0.15', @ode_isol2ep, ode_args{:}, p0, cont_args{:});
>> p0(7) = 2.0;
>> coco(prob, 'p7=2.0', @ode_isol2ep, ode_args{:}, p0, cont_args{:});
```

Here, the `'NSA'` setting of the `'ep'` toolbox is set to the non-default value of true, in order to ensure that neutral saddles are detected and located during continuation.

We may continue the family of Hopf bifurcations based at the second Hopf bifurcation point found during continuation with $p_7 = 0.4$, as suggested by the construction below.

```
>> bd = coco_bd_read('p7=0.4');
>> labs = coco_bd_labs(bd, 'HB');
>> prob = coco_prob();
>> prob = coco_set(prob, 'cont', 'PtMX', 50);
>> prob = ode_HB2HB(prob, '', 'p7=0.4', labs(2));
```

For non-degenerate Hopf bifurcation points, the super- or subcritical nature of the bifurcation (i.e., the orientation of the family of periodic orbits emanating from the bifurcation point) is determined by the sign of the first Lyapunov coefficient. Specifically, let v and w denote eigenvectors of $A := \partial_x F(x,p)$ and its transpose, respectively, corresponding to the eigenvalues $\mathrm{i}\omega$ and $-\mathrm{i}\omega$, such that $v^{*T} \cdot v = w^{*T} \cdot v = 1$. Furthermore, let B denote the 3-tensor, such that the i-th component of $B(a,b)$ is given by $a^T \cdot \partial_{xx} F_i(x,p) \cdot b$. For a quadratic vector field, the first Lyapunov coefficient is then given by the real part of

$$\frac{1}{2\omega} w^{*T} \cdot \left(B\big(v^*, (2\mathrm{i}\omega I_n - A)^{-1} \cdot B(v,v)\big) - 2B\big(v, A^{-1} \cdot B(v, v^*)\big)\right), \tag{19}$$

where * denotes complex conjugation. We encode the computation of the first Lyapunov coefficient in the COCO-compatible function `lyapunov` in the program file `lyapunov.m`, shown

below.

```
function [data y] = lyapunov(prob, data, u)

x = u(data.x_idx);
p = u(data.p_idx);

A = bykov_dx(x,p);
[X D]   = eig(A);
[m idx] = min(abs(real(diag(D))));
v  = X(:,idx);
om = imag(D(idx,idx));
vb = conj(v);
if m>1e-6
  y = NaN;
  return
end

[X D]   = eig(A');
[m idx] = min(abs(real(diag(D))));
w = X(:,idx);

if om*imag(D(idx,idx))>0
  w = conj(w);
end
w = w/conj(w'*v);

B  = bykov_dxdx(x,p);
B1 = zeros(numel(x),1);
B3 = zeros(numel(x),1);
for i=1:numel(x)
  Bmat  = reshape(B(i,:,:),[numel(x),numel(x)]);
  B1(i) = v.'*Bmat*v;
  B3(i) = v.'*Bmat*vb;
end
t1 = (2*sqrt(-1)*om*eye(numel(x))-A)\B1;
t2 = A\B3;
B2 = zeros(numel(x),1);
B4 = zeros(numel(x),1);
for i=1:numel(x)
  Bmat  = reshape(B(i,:,:),[numel(x),numel(x)]);
  B2(i) = vb.'*Bmat*t1;
  B4(i) = v.'*Bmat*t2;
end

y = real(w'*B2-2*w'*B4)/2/om;

end
```

The third input argument is here assumed to contain an array of numerical values for the problem variables and problem parameters, indexed by the `x_idx` and `p_idx` fields, respectively, of the function data structure. The call to the function `bykov_dxdx`, whose encoding in the program file `bykov_dxdx.m` is shown below, returns a three-dimensional array whose (i, j, k)-th entry equals the second partial derivative $\partial^2 F_i(x, p)/\partial x_j \partial x_k$.

```
function J = bykov_dxdx(x, p)

p1 = p(1,:);
p3 = p(3,:);
p5 = p(5,:);

J = zeros(3,3,3,numel(p1));

J(1,1,1,:) = 4*p1 - 4*p5;
J(1,1,2,:) = 4*p1 - p3;
J(1,1,3,:) = 4*p1;
J(1,2,1,:) = 4*p1 - p3;
J(1,2,2,:) = 4*p1;
J(1,2,3,:) = 4*p1;
J(1,3,1,:) = 4*p1;
J(1,3,2,:) = 4*p1;
J(1,3,3,:) = 4*p1;
J(2,1,2,:) = -p3;
J(2,2,1,:) = -p3;

end
```

We can now append the corresponding nonembedded monitor function to the continuation problem and introduce a special point associated with a zero crossing of the value of the first Lyapunov coefficient, corresponding to a generalized Hopf bifurcation.

```
>> [data uidx] = coco_get_func_data(prob, 'ep', 'data', 'uidx');
>> prob = coco_add_func(prob, 'lyap', @lyapunov, data.ep_eqn, ...
     'regular', 'L1', 'uidx', uidx);
>> prob = coco_add_event(prob, 'GH', 'L1', 0);
>> coco(prob, 'HB-curve', [], 1, {'p2' 'p7'}, [0.4 3]);
```

Exercises

1. Verify that the default value of the `'NSA'` setting of the `'ep'` toolbox implies that neutral saddles are not detected during continuation.

2. The `'HB-curve'` run includes the detection and location of several points with point type `'BTP'`. Verify that these separate portions of the solution manifold corresponding to Hopf bifurcations and neutral saddles, respectively.

3. Perform continuation under simultaneous variations in p_2 and p_7 along the family of saddle-node bifurcations based at one of the bifurcation points detected and located for $p_7 = 0.4$. Include monitoring and detection of zero crossings of the quadratic normal–form coefficient given by

$$\frac{1}{2}w^T \cdot B(v,v), \tag{20}$$

where v is a unit nullvector of the Jacobian $A = \partial_x F(x,p)$, w is a nullvector of A^T such that $w^T \cdot v = 1$, and B is defined as above. What are the points of intersection of this curve with the curve of Hopf bifurcations computed above?

4. The general formula for the first Lyapunov coefficient includes terms associated with the third partial derivatives of the vector field with respect to the state. Implement the appropriate modifications to the `lyapunov` function and apply this to the determination of the super- or subcritical nature of Hopf bifurcations in one of the dynamical systems used to illustrate the theory in "Numerical Methods for the Generalized Hopf Bifurcation," by Govaerts, W., Kuznetsov, Yu.A., and Sijnave, B., *SIAM Journal of Numerical Analysis*, 38(1), pp. 329-346, 2000.

---

7 Isola curves – **isola**

Closed curves of equilbria in the combined space of problem variables and subsets of problem parameters are known as isolas. These may be tracked under variations in other problem parameters by simultaneous continuation of a discrete number of equilibria along such a curve, together with conditions that the corresponding interpolating polygon approximates the isola.

Consider the vector field

$$F : (x,p) \mapsto \begin{pmatrix} -u + \lambda\tau(1-u)e^v \\ -v + 8\lambda\tau(1-u)e^v - \tau v \end{pmatrix} \tag{21}$$

describing chemical reactions in a continuous stirred tank reactor[9] and expressed in terms of the vector of problem variables $x = (u,v) \in \mathbb{R}^2$ and the vector of problem parameters $p = (\tau,\lambda) \in \mathbb{R}^2$. Vectorized encodings of the vector field and its Jacobians with respect to x and p are given in the functions `cstr`, `cstr_dx`, and `cstr_dp` shown below.

```
function f = cstr(x, p)

u = x(1,:);
v = x(2,:);
t = p(1,:);
l = p(2,:);

z = exp(v);

f = [-u + l.*t.*(1-u).*z; -v + 8*l.*t.*(1-u).*z-t.*v];

end
```

[9]See related analysis in "On the Numerical Continuation of Isolas of Equilibria," by Avitabile, Desroches, and Rodriquez, *International Journal of Bifurcation and Chaos*, 22(11), art. no. 1250277, 2012.

```
function J = cstr_dx(x, p)

u = x(1,:);
v = x(2,:);
t = p(1,:);
l = p(2,:);

z = exp(v);
J = zeros(2,2,numel(z));

J(1,1,:) = -1 - l.*t.*z;
J(1,2,:) = l.*t.*(1-u).*z;
J(2,1,:) = -8*l.*t.*z;
J(2,2,:) = -1+8*l.*t.*(1-u).*z-t;

end


function J = cstr_dp(x, p)

u = x(1,:);
v = x(2,:);
t = p(1,:);
l = p(2,:);

z = exp(v);
J = zeros(2,2,numel(z));

J(1,1,:) = l.*(1-u).*z;
J(1,2,:) = t.*(1-u).*z;
J(2,1,:) = 8*l.*(1-u).*z - v;
J(2,2,:) = 8*t.*(1-u).*z;

end
```

Continuation along a family of equilibria under variations in τ, as shown below, produces a sequence of solution points on an apparently closed curve.

```
>> ode_fcns  = {@cstr, @cstr_dx, @cstr_dp};
>> ode_args  = {ode_fcns{:}, [0.75 4], {'tau' 'lambda'}, [0.5; 0.11]};
>> cont_args = {1, 'tau', [0 2]};
>> bd1  = coco('initial', @ode_isol2ep, ode_args{:}, cont_args{:});
```

As the default atlas algorithm double-covers portions of the solution manifold, we identify an initial approximating polygon by the sequence of solution points starting with the second Hopf bifurcation point and ending with the third Hopf bifurcation point located during continuation (since these coincide). For each such polygon with N distinct vertices, we associate a parameter value s_i to the i-th vertex, such that $0 = s_1 < \cdots < s_{N+1} = L$ and

$$s_i - s_{i-1} = \sqrt{100(u_i - u_{i-1})^2 + (v_i - v_{i-1})^2 + 42.25(\tau_i - \tau_{i-1})^2}. \tag{22}$$

Linear interpolation between the polygonal vertices then assigns a unique value of s to each point on the polygon.

```
>> idxs = coco_bd_idxs(bd1, 'HB');
>> vars = coco_bd_col(bd1, 'x');
>> pars = coco_bd_col(bd1, {'tau' 'lambda'});
>> weights = [10; 1; 6.5; 1];
>> s = 0;
>> for idx = idxs(2)+1:idxs(3)
     dw = [vars(:,idx); pars(:, idx)] - [vars(:,idx-1); pars(:, idx-1)];
     dw = dw.*weights;
     s = [s; s(end) + norm(dw)];
   end
```

We proceed to construct an instance of a Hopf bifurcation continuation problem, with initial solution guess given by the second Hopf bifurcation point located above, together with $N-1$ instances of a regular equilibrium continuation problem, with initial solution guesses sampled uniformly in s along the approximating polygon found above.

```
>> labs = coco_bd_labs(bd1, 'HB');
>> prob = coco_prob();
>> prob = ode_HB2HB(prob, 'isola1', 'initial', '', labs(2));
>> N = 50;
>> vars = interp1(s, vars(:,idxs(2):idxs(3))', 0:s(end)/N:s(end))';
>> pars = interp1(s, pars(:,idxs(2):idxs(3))', 0:s(end)/N:s(end))';
>> prob = coco_set(prob, 'ep', 'SN', 'off', 'HB', 'off');
>> for idx = 2:N
     x0   = vars(:,idx);
     p0   = pars(:,idx);
     oid  = sprintf('isola%d', idx);
     prob = ode_isol2ep(prob, oid, @cstr, @cstr_dx, @cstr_dp, x0, p0);
   end
```

The corresponding composite continuation problem has dimensional deficit $2N-3$, since the dimensional deficit of the Hopf bifurcation continuation problem is -1 and each call to `ode_isol2ep` constructs an equilibrium continuation problem with dimensional deficit of 2. We reduce the dimensional deficit to $N-2$ by introducing $N-1$ gluing conditions that constrain all redundant copies of λ to equal the instance associated with the Hopf bifurcation point. We impose an additional N conditions on the collection of continuation variables by requiring that $s_i - s_{i-1} = \ell$ for all $i = 2, \ldots, N+1$ for some unknown variable ℓ.

```
>> [data uidx] = coco_get_func_data(prob, 'isola1.ep', 'data', 'uidx');
>> varidx = uidx(data.ep_eqn.x_idx);
>> paridx = uidx(data.ep_eqn.p_idx);
>> for idx = 2:N
     fid    = sprintf('isola%d.ep', idx);
     [data uidx] = coco_get_func_data(prob, fid, 'data', 'uidx');
     varidx = [varidx uidx(data.ep_eqn.x_idx)];
     paridx = [paridx uidx(data.ep_eqn.p_idx)];
   end
>> prob = coco_add_glue(prob, 'glue', paridx(2,1:end-1), paridx(2,2:end));
>> uidx = [varidx; paridx(1,:)];
>> prob = coco_add_func(prob, 'dist', @wdist, ...
     struct('w', repmat(weights(1:3), [1 N])), 'zero', ...
     'uidx', uidx, 'u0', s(end)/N);
```

Here, the function `wdist` is implemented in the COCO-compatible encoding shown below.

```
function [data y] = wdist(prob, data, u)

np = (numel(u)-1)/3;
pt = reshape(u(1:end-1), [3 np]);
ds = repmat(u(end), [np 1]);

dw = (pt - circshift(pt,[0 -1])).*data.w;
y  = sqrt(sum(dw.^2, 1))' - ds;

end
```

Finally, we introduce a function monitoring the value of ℓ and associate this with the initially inactive continuation parameter `'L'`. Continuation along a one-dimensional family of isolas then results by releasing the continuation parameters `'L'`, `'lambda'` and `'tau'`.

```
>> uidx = coco_get_func_data(prob, 'dist', 'uidx');
>> prob = coco_add_pars(prob, 'length', uidx(end), 'L');
>> prob = coco_set(prob, 'cont', 'PtMX', [20 50]);
>> bd2  = coco(prob, 'isola', [], 1, {'L' 'lambda' 'tau'}, [0 1]);
```

We may visualize the result of continuation by extracting the polygonal vertices from individual solution files, as shown in the following sequence of commands.

```
>> figure(1); clf; hold on; grid on; axis([0 1.5 0.2 1])
>> labs = coco_bd_labs(bd2);
>> for lab=labs
     var = zeros(N+1,4);
     for i=1:N
       sol = ep_read_solution(sprintf('isola%d', i), 'isola', lab);
       var(i,:) = [sol.x; sol.p]';
     end
     var(N+1,:) = var(1,:);
     plot(var(:,3), var(:,1),'r')
     plot(var(1,3), var(1,1),'ko')
     drawnow
   end
>> hold off
```

The `'ep'` utility `ep_read_solution` extracts a solution structure whose `x` and `p` field contain the values of the problem variables and problem parameters, respectively.

Exercises

1. Modify the plotting of individual isolas to distinguish between curve segments of stable and unstable equilibria, respectively.

2. Perform continuation of one of the Hopf bifurcation points found during the initial run under simultaneous variation in τ and λ and graph the corresponding solution curve

on top of the family of isolas.

3. Experiment with different values of the weights appearing under the radical in Eq. (22) and the number N of polygonal vertices and comment on the corresponding convergence and accuracy.

4. As an alternative to anchoring the approximating polygons on a Hopf bifurcation, consider imposing the phase condition

$$\sum_{i=2}^{N+1} \left((u_i^* - u_{i-1}^*)(u_i - u_{i-1}^*) + (v_i^* - v_{i-1}^*)(v_i - v_{i-1}^*) + (\tau_i^* - \tau_{i-1}^*)(\tau_i - \tau_{i-1}^*) \right) = 0 \quad (23)$$

on the family of polygonal vertices. Here, the * denotes a reference polygon, e.g., one obtained in a previous continuation step. Make the appropriate changes to the MATLAB script and comment on differences in interpretation and execution.

---

8 Optimization – `cusp_optim`

Consider the problem of finding stationary points of the function $(x, \kappa, \lambda) \mapsto \kappa$ along the manifold of equilibria for the cusp normal form

$$\dot{x} = \kappa - x\left(\lambda - x^2\right) \quad (24)$$

in terms of the scalar problem variable $x \in \mathbb{R}$ and vector of problem parameters $p = (\kappa, \lambda) \in \mathbb{R}^2$. In this case, $\kappa = x(\lambda - x^2)$ along the entire manifold and, consequently, stationary points occur wherever $x = \lambda - 3x^2 = 0$, i.e., for $x = \kappa = \lambda = 0$.

Alternatively, consider the Lagrangian

$$L(x, \kappa, \lambda, \mu_\kappa, \mu_\lambda, \ell_{\text{eq}}, \eta_\kappa, \eta_\lambda) = \mu_\kappa + \ell_{\text{eq}}\left(\kappa - x\left(\lambda - x^2\right)\right) + \eta_\kappa(\kappa - \mu_\kappa) + \eta_\lambda(\lambda - \mu_\lambda) \quad (25)$$

in terms of the continuation parameters μ_κ and μ_λ, and the Lagrange multipliers ℓ_{eq}, η_κ, and η_λ. Necessary conditions for stationary points along the constraint manifold correspond to points $(x, \kappa, \lambda, \mu_\kappa, \mu_\lambda, \ell_{\text{eq}}, \eta_\kappa, \eta_\lambda)$ for which $\delta L = 0$ for any infinitesimal variations δx, $\delta\kappa$, $\delta\lambda$, $\delta\mu_\kappa$, $\delta\mu_\lambda$, $\delta\ell_{\text{eq}}$, $\delta\eta_\kappa$, and $\delta\eta_\lambda$. In this case, these conditions take the form

$$\kappa - x\left(\lambda - x^2\right) = 0,\ \kappa - \mu_\kappa = 0,\ \lambda - \mu_\lambda = 0, \quad (26)$$

$$\ell_{\text{eq}}(3x^2 - \lambda) = 0,\ \ell_{\text{eq}} + \eta_\kappa = 0,\ -x\ell_{\text{eq}} + \eta_\lambda = 0, \quad (27)$$

$1 - \eta_\kappa = 0$, and $\eta_\lambda = 0$. The unique solution to these conditions is the point $x = \kappa = \lambda = \mu_\kappa = \mu_\lambda = \eta_\lambda = 0$, $\ell_{\text{eq}} = -1$, and $\eta_\kappa = 1$.

Stationary points along the solution manifold may be located using a method of staged continuation applied to the extended continuation problem obtained by combining (26) and

(27) with $\eta_\kappa - \nu_\kappa = 0$ and $\eta_\lambda - \nu_\lambda = 0$ in terms of the continuation variables $(x, \kappa, \lambda, \ell_{\text{eq}}, \eta_\kappa, \eta_\lambda)$ and continuation parameters $(\mu_\kappa, \mu_\lambda, \nu_\kappa, \nu_\lambda)$. The dimensional deficit of this extended continuation problem equals 2. We get one-dimensional solution manifolds by designating one of the continuation parameters as inactive.

Suppose, for example, that μ_κ, ν_κ, and ν_λ are active and μ_λ is inactive. Solutions of the form $(x, \kappa, \lambda, \mu_\kappa, \mu_\lambda, \ell_{\text{eq}}, \eta_\kappa, \eta_\lambda, \nu_\kappa, \nu_\lambda)$ to the corresponding restricted continuation problem are located on the three one-dimensional manifolds

$$\big(x, x(\mu_\lambda - x^2), \mu_\lambda, x(\mu_\lambda - x^2), \mu_\lambda, 0, 0, 0, 0, 0\big) \tag{28}$$

and

$$\left(\pm\frac{\sqrt{\mu_\lambda}}{\sqrt{3}}, \pm\frac{2\mu_\lambda\sqrt{\mu_\lambda}}{3\sqrt{3}}, \mu_\lambda, \pm\frac{2\mu_\lambda\sqrt{\mu_\lambda}}{3\sqrt{3}}, \mu_\lambda, \ell_{\text{eq}}, -\ell_{\text{eq}}, \pm\frac{\ell_{\text{eq}}\sqrt{\mu_\lambda}}{\sqrt{3}}, -\ell_{\text{eq}}, \pm\frac{\ell_{\text{eq}}\sqrt{\mu_\lambda}}{\sqrt{3}}\right), \tag{29}$$

parameterized by x and ℓ_{eq}, respectively. The manifolds in (29) intersect the manifold in (28) at the points

$$\left(\pm\frac{\sqrt{\mu_\lambda}}{\sqrt{3}}, \pm\frac{2\mu_\lambda\sqrt{\mu_\lambda}}{3\sqrt{3}}, \mu_\lambda, \pm\frac{2\mu_\lambda\sqrt{\mu_\lambda}}{3\sqrt{3}}, \mu_\lambda, 0, 0, 0, 0, 0\right), \tag{30}$$

corresponding to local extrema in the value of κ along the first manifold.

Notably, there is a unique point on each of the latter manifolds at which $\eta_\kappa = 1$. If we consider the restricted continuation problem obtained with μ_κ, μ_λ, and ν_λ active and ν_κ inactive and equal to 1, then solutions are located on the one-dimensional manifold

$$\big(x, 2x^3, 3x^2, 2x^3, 3x^2, -1, 1, -x, 1, -x\big) \tag{31}$$

parameterized by x. This manifold intersects the manifolds in (29) at the points

$$\left(\pm\frac{\sqrt{\mu_\lambda}}{\sqrt{3}}, \pm\frac{2\mu_\lambda\sqrt{\mu_\lambda}}{3\sqrt{3}}, \mu_\lambda, \pm\frac{2\mu_\lambda\sqrt{\mu_\lambda}}{3\sqrt{3}}, \mu_\lambda, -1, 1, \mp\frac{\sqrt{\mu_\lambda}}{\sqrt{3}}, 1, \mp\frac{\sqrt{\mu_\lambda}}{\sqrt{3}}\right). \tag{32}$$

Notably, the point along the tertiary manifold in (31) with $\eta_\lambda = 0$ coincides with the unique stationary point found previously.

We proceed to implement the extended continuation problem in COCO using the appropriate `'ep'` toolbox constructors. We encode the vector field and its derivatives in the `'ode'` compatible functions below.

```
function f = cusp(x,p)
f = p(1)-x*(p(2)-x^2);
end


function dfdx = cusp_dx(x,p)
dfdx = 3*x^2-p(2);
end
```

```
function dfdp = cusp_dp(x,p)
dfdp = [1 -x];
end


function dfdxdx = cusp_dxdx(x,p)
dfdxdx = 6*x;
end


function dfdxdp = cusp_dxdp(x,p)

dfdxdp = zeros(1,1,2);
dfdxdp(1,1,2) = -1;

end


function dfdpdp = cusp_dpdp(x,p)
dfdpdp = zeros(1,2,2);
end
```

In the first stage of construction, we use the `ode_isol2ep` toolbox constructor to encode the constraint conditions (26), as shown in the sequence of commands below.

```
>> prob  = coco_prob();
>> prob  = coco_set(prob, 'ode', 'vectorized', false);
>> fcns  = {@cusp, @cusp_dx, @cusp_dp, @cusp_dxdx, @cusp_dxdp, @cusp_dpdp};
>> prob1 = ode_isol2ep(prob, '', fcns{:}, 0, {'ka' 'la'}, [0; 0.5]);
```

Here `'ka'` and `'la'` represent the continuation parameters μ_κ and μ_λ, respectively. The adjoint conditions (27) are appended to the continuation problem using the `adjt_isol2ep` constructor, as shown below.

```
>> prob1 = adjt_isol2ep(prob1, '');
```

This call initializes all Lagrange multipliers at 0 and introduces the continuation parameters `'d.ka'` and `'d.la'` corresponding to ν_κ and ν_λ, respectively.

The first stage of continuation is now realized using the following call to the `coco` entry-point function.

```
>> bd1 = coco(prob1, 'cusp1', [], 1, {'ka' 'd.ka' 'd.la'}, [-0.5 0.5]);
```

We can switch to a secondary branch at either branch point located during this stage. In the code shown below, in the second stage of continuation, we continue from the second branch point until `'d.ka'` equals 1.

```
>> BPlab = coco_bd_labs(bd1, 'BP');
>> prob2 = ode_BP2ep(prob, '', 'cusp1', BPlab(1));
>> prob2 = adjt_BP2ep(prob2, '', 'cusp1', BPlab(1));
>> cont_args = {1, {'d.ka' 'ka' 'd.la'}, {[0 1] [-0.5 0.5]}};
>> bd2 = coco(prob2, 'cusp2', [], cont_args{:});
```

The third, and final, stage of continuation results from the next sequence of commands:

```
>> lab = coco_bd_labs(bd2, 'EP');
>> prob3 = ode_ep2ep(prob, '', 'cusp2', lab(2));
>> prob3 = adjt_ep2ep(prob3, '', 'cusp2', lab(2));
>> prob3 = coco_add_event(prob3, 'OPT', 'd.la', 0);
>> cont_args = {1, {'d.la' 'ka' 'la'}, {[], [-0.5 0.5], [-2 2]}};
>> coco(prob3, 'cusp3', [], cont_args{:});
```

Here, every special point with `'d.la'` equal to 0 that is detected during continuation is assigned the `'OPT'` label. We visualize the results of this staged approach to locating stationary points using the following commands:

```
>> figure(1); clf; hold on
>> coco_plot_bd('cusp1', 'ka', 'la', 'x')
>> thm = struct();
>> thm.ustab   = '';
>> thm.lspec   = {'g-', 'LineWidth', 1};
>> thm.special = {'OPT'};
>> thm.OPT     = {'kp', 'MarkerFaceColor', 'r', 'MarkerSize', 8};
>> coco_plot_bd(thm, 'cusp3', 'ka', 'la', 'x')
>> hold off; grid on; view(3)
```

We use a theme structure in the call to `ode_plot_bd` to override the default `'ep'` theme, since the tertiary solution manifold consists of approximations to saddle-node bifurcations of near-critical stability.

Exercises

1. Explain why the manifold in (31) consists of saddle-node bifurcations of the cusp normal form.

2. Explain why stationary points of the function $(x, \kappa, \lambda) \mapsto \kappa$ on the manifold of equilibria for the cusp normal form may be found from consideration of the Lagrangian

$$L(x, \kappa, \lambda, \ell_{\text{eq}}) = \kappa + \ell_{\text{eq}}(\kappa - x\left(\lambda - x^2\right))$$

   What are the advantages and disadvantages of this formulation compared to (25) for the theoretical analysis? What about for the method of staged continuation?

3. Consider the problem of finding stationary points of the function $(x, \kappa, \lambda) \mapsto \lambda$ along the manifold of equilibria for the cusp normal form. Repeat the analysis in this section and verify your theoretical predictions using COCO.

4. Find stationary points of the function $(x, y, z) \mapsto x$ along the manifold of solutions to the algebraic equation

$$(r^2 + 2y - 1)\left((r^2 - 2y - 1)^2 - 8z^2\right) + 16xz(r^2 - 2y - 1) = 0,$$

   where $r^2 = x^2 + y^2 + z^2$, and verify your predictions using staged continuation in COCO.

---

9 Toolbox reference

The toolbox constructors implement zero and monitor functions appropriate to the nature of the continuation problem and the detection of special points along the solution manifold. Event handlers ensure that solution data specifically associated with special points is appropriately stored to disk.

9.1 Zero problems

For continuation of general equilibria, the zero problem is given in terms of the vector of continuation variables $u = (x, p)$ by $\Phi(u) = 0$, where $\Phi : u \mapsto F(x, p)$ is the corresponding family of zero functions. Its dimensional deficit equals the number q of problem parameters. Simultaneous continuation of arrays of perturbations and their images under the Jacobian $\partial_x F(x, p)$ is accomplished using the composite zero problem $\Phi(u) = 0$, where $u = (x, p, \mathfrak{vec}(v), \mathfrak{vec}(w))$, in terms of the vectorization operator $\mathfrak{vec}$, and

$$\Phi : u \mapsto \begin{pmatrix} F(x,p) \\ \partial_x F(x,p) \cdot v - w \end{pmatrix} \tag{33}$$

is the corresponding family of zero functions. Its dimensional deficit equals $q + nm$, where m equals the number of columns of v.

In the current implementation of the `'ep'` toolbox, the zero problem for continuation of saddle-node bifurcation points is given in terms of the vector of continuation variables $u = (x, p, v, w)$ by $\Phi(u) = 0$, where

$$\Phi : u \mapsto \begin{pmatrix} F(x,p) \\ \partial_x F(x,p) \cdot v - w \\ w \\ v^T \cdot v - 1 \end{pmatrix} \tag{34}$$

is the corresponding family of zero functions. Its dimensional deficit equals $q - 1$.

Finally, in the current implementation of the `'ep'` toolbox, the zero problem for continuation of Hopf bifurcation points is given in terms of the vector of continuation variables $u = (x, p, v, \tilde{v}, w, k)$ by $\Phi(u) = 0$, where

$$\Phi : u \mapsto \begin{pmatrix} F(x,p) \\ \partial_x F(x,p) \cdot v - \tilde{v} \\ \partial_x F(x,p) \cdot \tilde{v} - w \\ kv + w \\ v^T \cdot v - 1 \\ n^T \cdot v \end{pmatrix} \tag{35}$$

is the corresponding family of zero functions. Its dimensional deficit equals $q-1$. The vector n is updated before each continuation step by normalizing the vector

$$(-\tilde{v}^T \cdot v)v + (v^T \cdot v)\tilde{v}. \tag{36}$$

9.2 Calling syntax

The calling syntax for a generic 'ep' toolbox constructor tbc_ctr is of the form

```
prob = tbx_ctr(prob, oid, varargin)
```

where prob denotes a (possibly empty) continuation problem structure and oid is a string representing an object instance identifier.

In the case of the ode_isol2ep toolbox constructor, the varargin input argument adheres to the following syntax

```
varargin = fcns x0  [pnames]  p0 [opts]
```

where

```
fcns = @f  [@dfdx  [@dfdp [@dfdxdx [@dfdxdp [@dfdpdp]]]]]
```

Here, @f denotes a required function handle to the encoding of the operator F, and each of the optional arguments @dfdx, @dfdp, @dfdxdx, @dfdxdp, and @dfdpdp is either an empty array ([]) or a function handle to the corresponding array of partial derivatives with respect to the state variables and problem parameters, respectively. Notably, if adjoint equations are to be constructed using the adjt_isol2ep constructor, then the preceding call to ode_isol2ep must include explicit function handles to encodings of the Jacobians with respect to x and p, respectively.

An initial solution guess for the problem variables and problem parameters is given by the x0 and p0 input arguments, respectively. An optional designation of string labels for continuation parameters assigned to track the problem parameters is provided with pnames, which is either a single string or a cell array of strings. An error is thrown if the number of string labels in this optional argument, when present, differs from the number of elements of p0. The optional opts argument is either of the strings '-ep-end' and '-end-ep', indicating the end of input to the ode_isol2ep toolbox constructor, or the string '-var' followed by a numerical matrix with n rows, indicating the simultaneous continuation of arrays of perturbations and their images under the Jacobian $\partial_x F(x, p)$. In the latter case, the numerical matrix constitutes an initial solution guess for the variable v in (33).

For each of the ode_ep2ep, ode_BP2ep, ode_HB2HB, and ode_SN2SN toolbox constructors, the varargin input argument adheres to the syntax

```
varargin = run [soid] lab [opts]
```

Here, run denotes a string identifying a previous run and lab is an integer identifying the corresponding solution file. The optional argument soid denotes a source object instance identifier, in the case that this differs from oid. In all cases, the optional opts argument may equal either of the strings '-ep-end' and '-end-ep', thereby denoting explicitly the end of the sequence of arguments to an 'ep' toolbox constructor. For ode_ep2ep, opts may also contain the string '-switch', which, when present, implies that continuation should proceed along a secondary solution branch through the given solution. For ode_ep2ep and ode_BP2ep, opts may also contain the string '-var' followed by a numerical matrix with n

rows, indicating the simultaneous continuation of arrays of perturbations and their images under the Jacobian $\partial_x F(x,p)$. In the latter case, the numerical matrix constitutes an initial solution guess for the variable v in (33).

9.3 Adjoint functions

For continuation of general equilibria, the contributions to the adjoint equations associated with variations in x and p are expressed in terms of the Jacobians $\partial_x F(x,p)$ and $\partial_p F(x,p)$ and a subset of components of the vector of continuation multipliers λ. The appropriate changes to the continuation problem structure are invoked using the `adjt_isol2ep` constructor, following a preceding call to the `ode_isol2ep` constructor that includes function handles to explicit encodings of these Jacobians. Specifically, in the call

```
prob = adjt_isol2ep(prob, oid)
```

the `oid` argument denotes an object identifier associated with the toolbox instance created by the preceding call to `ode_isol2ep`. The corresponding components of λ are initialized to 0.

If the preceding call to `ode_isol2ep` includes an explicit list of parameter labels, then the corresponding additions to the adjoint equations are automatically encoded by the call to `adjt_isol2ep`. The corresponding components of the vector of continuation multipliers η are initialized to 0.

In a similar fashion, a call to `ode_ep2ep` or `ode_BP2ep` may be followed by a call to `adjt_ep2ep` or `adjt_BP2ep`, respectively, with identical arguments, in order to append the contributions to the adjoint equations associated with the reconstructed continuation problem. In either case, the associated elements of the vectors of continuation multipliers λ and η are automatically initialized from the corresponding values stored in a solution file.

9.4 Continuation parameters and toolbox settings

The inclusion of the `pnames` optional argument in the call to the `ode_isol2ep` toolbox constructor ensures the encoding in the continuation problem structure of initially inactive, embedded continuation parameters equal in number to the number of string labels (which must equal the number of problem parameters). These string labels are stored in the function data structure, written to disk with each solution file, and reused in the event that a continuation problem is created from saved solution data using either of the remaining toolbox constructors. A subsequent call to `adjt_isol2ep`, `adjt_ep2ep`, or `adjt_BP2ep` ensures the encoding in the continuation problem structure of an accompanying set of initially inactive embedded continuation parameters corresponding to the associated subset of the vector of continuation multipliers η, and with labels obtained by appending `'d.'` to the original string labels.

If the `'bifus'` option of the `'ep'` toolbox is set to true (as it is by default), the `ode_isol2ep`, `ode_ep2ep`, and `ode_BP2ep` constructors also encode the three nonembedded continuation parameters `'OID.ep.test.HB'`, `'OID.ep.test.SN'`, and `'OID.ep.test.USTAB'`,

associated with detection of Hopf bifurcations/neutral saddle points and saddle-node bifurcations, and with monitoring the Lyapunov stability (the number of unstable eigenvalues) of the equilibrium point, respectively. In this case, changes to the sign of the first two of these continuation parameters trigger the detection of special points denoted by `'HB'` and `'SN'`, respectively. If the `'NSA'` option of the `'ep'` toolbox is set to true (it is false by default), then neutral saddles, denoted by `'NSA'`, are also located.

If the `'BTP'` option of the `'ep_HB'` toolbox is set to true (as it is by default), the `ode_HB2HB` constructor encodes the additional single nonembedded continuation parameter `'OID.ep.test.BT'` associated with the detection of Bogdanov-Takens bifurcation points. Changes to the sign of this continuation parameter trigger the detection of special points denoted by `'BTP'`.

To set options associated with a specific `'ep'` instance with object instance identifier `OID`, use the syntax

```
>> prob = coco_set(prob, 'OID.ep', ...
```

To set options associated with all `'ep'` instances whose object instance identifiers derive from a parent identifier `PID`, use the syntax

```
>> prob = coco_set(prob, 'PID.ep', ...
```

To set options for all `'ep'` instances in a continuation problem, use the syntax

```
>> prob = coco_set(prob, 'ep', ...
```

As explained in *Recipes for Continuation*, precedence is given to settings defined using the most specific path identifier. See the output of the `ep_settings` utility for a list of supported settings and their default or current values.

9.5 Toolbox output

By default, the bifurcation data cell array stored during continuation and returned by the `coco` entry-point function (given a receiving variable) includes four columns with headers `'OID.x'`, `'||OID.x||_2'`, `'MAX(OID.x)'`, and `'MIN(OID.x)'` with data given by the vector of problem variables, the corresponding Euclidean norm, and the maximum and minimum entries of this vector, respectively, and `OID` representing an object instance identifier (the period is omitted when `OID` equals the empty string). In addition, if eigenvalues of the Jacobian $\partial_x F(x,p)$ are computed during continuation, then these are included in a column with header `'OID.eigs'`. All continuation parameters are included in the bifurcation data cell array by default, but printed to screen during continuation only if included in the list of arguments to the `coco` entry-point function.

For general equilibrium points, the `sol` output argument of the `ep_read_solution` utility contains

- the vector of problem variables (in the `x` field),
- the vector of problem parameters (in the `p` field),

- the vector of continuation variables (in the `u` field),
- the tangent vector to the corresponding curve segment (in the `t` field).

If eigenvalues of the Jacobian $\partial_x F(x,p)$ are computed during continuation, then these are contained in the field `ep_test.la`. In the case of simultaneous continuation of arrays of perturbations and their images under the Jacobian $\partial_x F(x,p)$, the field `var.v` contains the array of perturbations.

For branch points (located by the atlas algorithm) the `t0` field contains a singular vector normal to `t`. For saddle-node bifurcation points, the field `var.v` contains the unit eigenvector v of $\partial_x F(x,p)$ corresponding to the zero eigenvalue. For Hopf bifurcation points, the `sol` output argument contains

- the square $k(=\omega^2)$ of the Hopf frequency (in the `hb.k` field),
- a unit eigenvector of the squared Jacobian $\partial_x F(x,p) \cdot \partial_x F(x,p)$ corresponding to the eigenvalue $-k$ (in the first column of the `var.v` field).

The `ep_plot_theme` toolbox utility defines the default visualization theme for the `'ep'` toolbox. The command

```
>> thm = ep_plot_theme('ep')
```

assigns the default theme for visualization of the results of continuation of general equilibrium points to the `thm` variable. Similarly, the commands

```
>> thm_SN = ep_plot_theme('ep.SN')
>> thm_HB = ep_plot_theme('ep.HB')
```

assign the default themes for continuation of saddle-node and hopf bifurcation points, respectively, to the variables `thm_SN` and `thm_HB`. Notably, when visualizing the results of continuation of general equilibria, the continuation parameter `'OID.ep.test.USTAB'` is used to distinguish branches of stable and unstable equilibria, respectively. In this case, to include markers identifying saddle-node bifurcations, Hopf bifurcations, or neutral saddles, the labels `'SN'`, `'HB'`, or `'NSA'` should be added to the `special` field of the problem-specific plotting theme. Similarly, to include markers identifying Bogdanov-Takens bifurcation points during continuation of Hopf bifurcations, the label `'BTP'` should be added to the `special` field of the problem-specific plotting theme.

9.6 Developer's interface

Continuation problems constructed with the `'ep'` toolbox constructors may be embedded in larger continuation problems that contain additional continuation variables, zero functions, and/or monitor functions. Each `'ep'` instance is associated with a toolbox instance identifier obtained by prepending an object instance identifier to the string `'ep'`.

The `coco_get_func_data` core utility may be used to extract the function dependency index set (the `'uidx'` option) and the toolbox data structure (the `'data'` option) associated

with the basic equilibrium problem. As shown in the examples and described further in the documentation of the `ep_add` interface function, the content of the `ep_eqn` field of the toolbox data structure includes context-independent arrays of integer indices for the vector of problem variables (`x_idx`) and problem parameters (`p_idx`), respectively. In the case of simultaneous continuation of arrays of perturbations and their images under the Jacobian $\partial_x F(x, p)$, the `ep_var` field of the toolbox data structures includes context-independent arrays of integer indices for the array of perturbations (`v_idx`) and their images (`w_idx`). In the case of continuation of saddle-node bifurcations, the `ep_sn` field of the toolbox data structure includes context-independent arrays of integer indices for the vectors v (`v_idx`) and w (`w_idx`). In the case of continuation of hopf bifurcations, the `ep_hb` field of the toolbox data structure includes context-independent arrays of integer indices for the vectors v (`v_idx`), w (`w_idx`), and k (`k_idx`), as well as the content of the vector n (`nv`).

The `coco_get_adjt_data` core utility may be used to extract the adjoint row (the `'afidx'` option) and column (the `'axidx'` option) index sets as well as the toolbox adjoint data structure (the `'data'` option). The content of the `ep_opt` field of the adjoint data structure includes context-independent arrays of integer indices for the columns associated with problem variables (`x_idx`) and problem parameters (`p_idx`), respectively.

The `'ep'` toolbox data structure contains several fields that are associated with the `'ode'` toolbox family. These include function handles to the vector field (`fhan`), to its Jacobians (`dfdxhan` and `dfdphan`), and to functions evaluating the second derivatives with respect to the state variables and problem parameters (`dfdxdxhan`, `dfdxdphan`, and `dfdpdphan`), a cell array of string labels for the continuation parameters associated with problem parameters (`pnames`), the state-space dimension (`xdim`), and the number of problem parameters (`pdim`).

The `'ep'` toolbox data structure contains a number of implementation-dependent internal fields whose use may change in the future. Accessing such internal fields is deprecated.