The Periodic Orbit Toolbox

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

Frank Schilder
Department of Mathematics
Technical University of Denmark

January 7, 2025

Contents

1 Introduction

The `'po'` toolbox is a basic toolbox for continuation along families of single-segment periodic orbits in smooth dynamical systems or multi-segment periodic orbits in hybrid dynamical systems for evolution equations of the form

$$\dot{x} = F(t, x, p), t \in [T_0, T_0 + T] \tag{1}$$

in terms of an initial time T_0, an interval length T, a vector of state variables $x \in \mathbb{R}^n$, a vector of problem parameters $p \in \mathbb{R}^q$, and a nonlinear operator $F : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^n$. For infinite-dimensional problems, the toolbox applies to suitable discretizations of x and F. The `'po'` toolbox belongs to the `'ode'` toolbox family, and is modeled on the `'po'` and `'hspo'` toolboxes, described in *Recipes for Continuation*[1].

The `'po'` toolbox relies on the `'coll'` toolbox for adaptive discretization of each trajectory segment. It supports autonomous implementations of the operator F that omit dependence on the first argument. In fact, unless otherwise indicated, this is the assumed default and explicit time-dependence must be indicated by an optional setting.

The `'po'` toolbox supports detection of branch and fold points (inherited from the associated atlas class), as well as critical thresholds associated with an estimated discretization error (inherited from the `'coll'` toolbox). In addition, the `'po'` toolbox supports detection of

- saddle-node (cyclic fold) bifurcations,
- period-doubling (flip) bifurcations
- Neimark-Sacker (torus) bifurcations, and
- neutral saddle points (optional and disabled by default),

as well as continuation along families of saddle-node, period-doubling, and Neimark-Sacker bifurcations. For continuation of periodic orbits, the `'po'` toolbox supports the construction of the associated adjoint equations[2,3].

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

- `ode_isol2po` for continuation along a family of single-segment periodic orbits in a smooth dynamical system from an initial solution guess;
- `ode_po2po` for continuation along a family of single-segment periodic orbits in a smooth dynamical system from a saved solution point;

[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," *SIAM J. Applied Dynamical Systems* **17(2)**, pp. 1117–1151, 2018.

[3] Li, M. & Dankowicz, H., "Optimization with Equality and Inequality Constraints Using Parameter Continuation," *Applied Mathematics and Computation* **375**, art. no. 125058, 2020.

- `ode_HB2po` for continuation along a family of single-segment periodic orbits in a smooth dynamical system emanating from a Hopf bifurcation point along a family of equilibria;
- `ode_PD2po` for continuation along a family of single-segment periodic orbits in a smooth dynamical system from a period-doubling bifurcation point along the bifurcated branch;
- `ode_isol2hspo` for continuation along a family of multi-segment periodic orbits in an autonomous hybrid dynamical system from an initial solution guess;
- `ode_hspo2hspo` for continuation along a family of multi-segment periodic orbits in an autonomous hybrid dynamical system from a saved solution point;
- `ode_PD2hspo` for continuation along a family of multi-segment periodic orbits in an autonomous hybrid dynamical system from a period-doubling bifurcation point along the bifurcated branch;
- `ode_SN2SN` for continuation along a family of saddle-node bifurcation points from a saved saddle-node bifurcation point;
- `ode_PD2PD` for continuation along a family of period-doubling bifurcation points from a saved period-doubling bifurcation point;
- `ode_TR2TR` for continuation along a family of Neimark-Sacker bifurcation points from a saved Neimark-Sacker bifurcation point.

The additional constructors `adjt_isol2po` and `adjt_po2po` contribute terms to the adjoint equations associated with the zero and monitor functions appended to a continuation problem by the `ode_isol2po` and `ode_po2po` constructors, respectively. Similarly, `adjt_isol2hspo` and `adjt_hspo2hspo` contribute terms to the adjoint equations associated with the zero and monitor functions appended to a continuation problem by the `ode_isol2hspo` and `ode_hspo2hspo` constructors, respectively.

Usage is illustrated in the following several examples. Each example corresponds to fully documented code in the `coco/po/examples` folder in the COCO release. Slight differences between the code included below and the example implementations in `coco/po/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 A forced harmonic oscillator – `linode`

Consider the nonautonomous dynamical system governed by the vector field

$$F(t,x,p) = \begin{pmatrix} x_2 \\ -x_2 - px_1 + \cos t \end{pmatrix} \tag{2}$$

in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and the scalar problem parameter $p \in \mathbb{R}$. For every p, there exists a unique periodic orbit given by

$$x_1(t) = \frac{\sin t + (p-1)\cos t}{p^2 - 2p + 2},\ x_2(t) = \frac{\cos t - (p-1)\sin t}{p^2 - 2p + 2} \tag{3}$$

with $\mathcal{L}_2$ norm

$$\|x(t)\|_{\mathcal{L}_2[0,2\pi]} = \sqrt{\frac{2\pi}{p^2 - 2p + 2}} \tag{4}$$

and Floquet multipliers

$$e^{(-1\pm\sqrt{1-4p})\pi}. \tag{5}$$

We encode vectorized implementations of the vector field and its Jacobians with respect to the state variables and parameters in the functions `linode`, `linode_DFDX`, `linode_DFDP`, and `linode_DFDT`, as shown below.

```
function y = linode(t, x, p)

x1 = x(1,:);
x2 = x(2,:);
p1 = p(1,:);

y(1,:) = x2;
y(2,:) = -x2-p1.*x1+cos(t);

end


function J = linode_DFDX(t, x, p)

x1 = x(1,:);
p1 = p(1,:);

J = zeros(2,2,numel(x1));
J(1,2,:) = 1;
J(2,1,:) = -p1;
J(2,2,:) = -1;

end


function J = linode_DFDP(t, x, p)

x1 = x(1,:);

J = zeros(2,1,numel(x1));
J(2,1,:) = -x1;

end


function Jt = linode_DFDT(t, x, p)

Jt = zeros(2,numel(t));
```

```
  Jt(2,:) = -sin(t);

  end
```

In the following commands, we assign the parameter label `'p'` to the `pnames` variable and the numerical value 1 to `p0` corresponding to the initial value for the problem parameter p.

```
>> pnames = 'p';
>> p0     = 1;
```

The following call to `ode45` then generates an initial solution guess for the discretization of a periodic orbit.

```
>> [t0 x0] = ode45(@(t,x) linode(t,x,p0), [0 2*pi], [0; 1]);
```

The following sequence of commands encodes a periodic orbit continuation problem using the `ode_isol2po` constructor.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'autonomous', false);
>> prob = coco_set(prob, 'coll', 'NTST', 15);
>> coll_func = {@linode, @linode_DFDX, @linode_DFDP, @linode_DFDT};
>> coll_args = [coll_func, {t0, x0, pnames, p0}];
>> prob = ode_isol2po(prob, '', coll_args{:});
```

Here the `'autonomous'` setting of the `'ode'` toolbox is set to false, to indicate the explicit dependence on the independent variable t. The number of discretization intervals used by the `'coll'` toolbox is assigned the initial value of 15.

We proceed to assign the integer 1 to the optional `'NAdapt'` setting of the atlas algorithm to ensure that adaptive changes are made to the orbit discretization after each successful step of continuation.

```
>> prob = coco_set(prob, 'cont', 'NAdapt', 1);
>> coco(prob, 'run', [], 1, 'p', [0.2 2]);
```

The dimensional deficit of the continuation problem is 0. Since the desired manifold dimensionality is 1, it follows that the continuation parameter `'p'` is released during continuation and allowed to vary on the interval $[0.2, 2]$.

We may restart continuation from one of the periodic orbits obtained in the previous run, as shown in the following commands.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'autonomous', false);
>> prob = ode_po2po(prob, '', 'run', 3);
>> prob = coco_set(prob, 'cont', 'NAdapt', 1);
>> coco(prob, 'run_again', [], 1, 'p', [0.05 3]);
```

These commands differ from the previous construction only in the use of the `ode_po2po` constructor.

Exercises

1. Use the `po_read_solution` utility to extract the state-space trajectory corresponding to one of the solutions found during continuation and graph this together with the theoretical prediction.

2. In this example, the `'coll'` toolbox stores the $\mathcal{L}_2$ norm associated with each solution trajectory in the `'||po.orb.x||_{L_2[0,T]}'` column of the bifurcation data cell array stored to disk during continuation. Use the `coco_bd_read` and `coco_bd_col` utilities to extract the corresponding numerical values from one of the continuation runs and graph their dependence on p together with the theoretical prediction.

3. Use the `po_read_solution` utility to extract the value of the problem parameter p and the corresponding Floquet multipliers from each stored solution file and compare to the theoretical prediction.

4. Repeat the analysis for the case of the harmonically excited linear oscillator

$$\dot{x}_1 = x_2,\ \dot{x}_2 = -x_2 - px_1 + \cos\omega t$$

under variations in the excitation frequency ω. Compare the numerical results to the corresponding theoretical predictions.

---

3 Nonlinear hardening – **bistable**

The nonautonomous dynamical system given by the vector field

$$F(t,x,p) = \begin{pmatrix} x_2 \\ -dx_2 - x_1 - x_1^3 + A\cos\left(2\pi t/T\right) \end{pmatrix} \tag{6}$$

in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and the vector of problem parameters $p = (T, A, d) \in \mathbb{R}^3$ represents the response of a hardening nonlinear oscillator to harmonic excitation with period T. In particular, larger excitation amplitudes are associated with the onset of *bistability*, i.e., intervals in excitation frequency with coexisting stable steady-state responses.

We encode the vector field and its Jacobians in the functions `bistable`, `bistable_dx`, `bistable_dp`, and `bistable_dt` shown below.

```
function y = bistable(t, x, p)

x1 = x(1,:);
x2 = x(2,:);
p1 = p(1,:);
p2 = p(2,:);
p3 = p(3,:);
```

```
y(1,:) = x2;
y(2,:) = -p3.*x2-x1-x1.^3+p2.*cos(2*pi./p1.*t);

end


function J = bistable_dx(t, x, p)

x1 = x(1,:);
p3 = p(3,:);

J = zeros(2,2,numel(x1));
J(1,2,:) = 1;
J(2,1,:) = -1-3*x1.^2;
J(2,2,:) = -p3;

end


function J = bistable_dp(t, x, p)

x2 = x(2,:);
p1 = p(1,:);
p2 = p(2,:);

J = zeros(2,3,numel(p1));
J(2,1,:) = 2*pi./p1.^2.*p2.*t.*sin(2*pi./p1.*t);
J(2,2,:) = cos(2*pi./p1.*t);
J(2,3,:) = -x2;

end


function Jt = bistable_dt(t, x, p)

p1 = p(1,:);
p2 = p(2,:);

Jt = zeros(2,numel(t));
Jt(2,:) = -2*pi*p2./p1.*sin(2*pi./p1.*t);

end
```

An approximate periodic orbit may now be obtained using the `ode45` MATLAB integrator.

```
>> p0 = [2*pi; 0.015; 0.04];
>> [~, x0] = ode45(@(t,x) bistable(t,x,p0), [0 500*pi], [0; 1]);
>> [t0 x0] = ode45(@(t,x) bistable(t,x,p0), [0 2*pi], x0(end,:)');
```

The following sequence of commands then encodes a periodic orbit continuation problem.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'autonomous', false);
>> funcs = {@bistable, @bistable_dx, @bistable_dp, @bistable_dt};
>> coll_args = [funcs, {t0, x0, {'T' 'A' 'd'}, p0}];
>> prob = ode_isol2po(prob, '', coll_args{:});
```

Notably, this does not recognize that the orbit period must equal an integer multiple of the excitation period T. To this end, the following commands constrain the interval length to equal the value of the first problem parameter.

```
>> [data uidx] = coco_get_func_data(prob, 'po.orb.coll', 'data', 'uidx');
>> maps = data.coll_seg.maps;
>> prob = coco_add_glue(prob, 'glue', uidx(maps.T_idx), uidx(maps.p_idx(1)));
```

The dimensional deficit of the continuation problem encoded thus far equals -1. Continuation under variations in the excitation period then proceeds by releasing both `'po.period'` and `'T'` and allowing these to vary, as shown below.

```
>> cont_args = {1, {'po.period' 'T'}, [2*pi/1.3 2*pi/0.7]};
>> bd1  = coco(prob, 'freq_resp', [], cont_args{:});
```

The saddle-node bifurcation points found during the previous continuation run may be used as starting points for continuation along a family of saddle-node bifurcations, as shown in the following sequence of commands.

```
>> labs = coco_bd_labs(bd1, 'SN');
>> prob = coco_prob();
>> prob = coco_set(prob, 'coll', 'NTST', 25);
>> prob = ode_SN2SN(prob, '', 'freq_resp', labs(1));
>> [data uidx] = coco_get_func_data(prob, 'po.orb.coll', 'data', 'uidx');
>> maps = data.coll_seg.maps;
>> prob = coco_add_glue(prob, 'glue', uidx(maps.T_idx), uidx(maps.p_idx(1)));
>> prob = coco_set(prob, 'cont', 'NAdapt', 5);
>> cont_args = {1, {'po.period' 'T' 'A'}, [2*pi/1.3 2*pi/0.7]};
>> bd2  = coco(prob, 'saddle-node', [], cont_args{:});
```

The fold observed along this family corresponds to a cusp bifurcation and the onset of bistability in the nonlinear frequency response of the hardening oscillator.

The *backbone curve* for the nonlinear oscillator is the one-dimensional family of periodic orbits obtained for $A = d = 0$ and emanating from the limit of zero response amplitude with period equal to 2π. The following sequence of commands construct a corresponding periodic orbit continuation problem.

```
>> t0 = (0:0.01:2*pi)';
>> x0 = 2e-2*[sin(t0) cos(t0)];
>> p0 = [2*pi; 0; 0];
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'autonomous', false);
>> prob = coco_set(prob, 'po', 'bifus', 'off');
>> funcs = {@bistable, @bistable_dx, @bistable_dp, @bistable_dt};
>> coll_args = [funcs, {t0, x0, {'T' 'A' 'd'}, p0}];
>> prob = ode_isol2po(prob, '', coll_args{:});
```

Notably, the value of T has no effect on this continuation problem, as long as $A = 0$, but must not equal 0 given the division with T encoded in `bistable.m`. Although the dimensional deficit of the continuation problem encoded thus far equals 0, this problem is degenerate, since arbitrary shifts in time are still solutions. We restrict attention to a particular phase

by holding fixed the value of x_1 on the initial point on the periodic orbit as shown below.

```
>> [data uidx] = coco_get_func_data(prob, 'po.orb.coll', 'data', 'uidx');
>> maps = data.coll_seg.maps;
>> prob = coco_add_pars(prob, 'section', uidx(maps.x0_idx(1)), 'y0');
>> prob = coco_set(prob, 'cont', 'NAdapt', 1);
>> cont_args = { 1, { 'po.period'  'd' }, [2*pi/1.3 2*pi/0.7] };
>> coco(prob, 'backbone', [], cont_args{:});
```

Note that both `'po.period'` and `'d'` are allowed to vary during continuation, thereby ensuring a dimensional deficit of 1 corresponding to a one-dimensional solution manifold. Nevertheless, the value of `'d'` remains approximately 0, since periodic orbits with nonzero amplitude exist for $A = 0$ only if $d = 0$.

Exercises

1. Use the `coco_add_event` utility to add special points associated with a sequence of values of A during the continuation along the family of saddle-node bifurcations. Then use the `ode_po2po` constructor to start continuation along a family of periodic orbits from each of the corresponding solutions.

2. Use the $\mathcal{L}_2[0,T]$ norm stored with the bifurcation data to graph the nonlinear response obtained in the previous exercises in the same diagram as the backbone as a function of the orbit period. Can you explain why the backbone appears to closely approximate the upper saddle-node bifurcation along the hardening response curves?

3. The fold along the family of saddle-node bifurcations corresponds to a cusp bifurcation. Approximate this bifurcation by three simultaneous instances of periodic orbits in the bistable case, at the same period of excitation and with similar amplitudes. Perform continuation of this configuration under simultaneous variation in the excitation period T, excitation amplitude A, and damping d.

4. Repeat the analysis in this section and the previous exercises for a softening nonlinear oscillator, for which the sign of the coefficient of the cubic term in the vector field in positive.

---

4 The Hopf normal form – `hopf`

Consider the autonomous dynamical system defined by the vector field

$$F(x,p) = \begin{pmatrix} x_1(p_1 + p_2 r^2 - r^4) - x_2 \\ x_2(p_1 + p_2 r^2 - r^4) + x_1 \end{pmatrix}, \tag{7}$$

where $r^2 = x_1^2 + x_2^2$, in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and the vector of problem parameters $p = (p_1, p_2) \in \mathbb{R}^2$. This corresponds to an unfolding of the normal form for a Hopf bifurcation of an equilibrium.

Equation (7) implies the existence of a unique equilibrium at the origin $(x_1, x_2) = (0, 0)$ for all values of p_1 and p_2. The equilibrium is asymptotically stable for $p_1 < 0$ and unstable for $p_1 > 0$. The critical value $p_1 = 0$ corresponds to a Hopf bifurcation, from which there emanates a locally unique family of periodic orbits of the form

$$x_1(t) = r^* \cos t,\ x_2 = r^* \sin t \tag{8}$$

with norm

$$\|x\|_2 = \frac{1}{2\pi} \int_0^{2\pi} \sqrt{x_1^2(t) + x_2^2(t)}\, \mathrm{d}t = r^* \tag{9}$$

and Floquet multiplier

$$e^{2\pi(p_1 + 3p_2 r^{*2} - 5r^{*4})}, \tag{10}$$

where $r^* > 0$ is given by

$$r^{*,2} = \frac{p_2 \pm \sqrt{p_2^2 + 4p_1}}{2}. \tag{11}$$

If $p_2 < 0$, the family of periodic orbits exists only for $p_1 > 0$. For each such value of p_1, the periodic orbit is asymptotically stable, and the Hopf bifurcation is *supercritical*. On the other hand, for $p_2 > 0$, the Hopf bifurcation is *subcritical*. In this case, there exists a unique asymptotically stable periodic orbit for $p_1 \geq 0$ and no periodic orbits for $p_1 < -p_2^2/4$. For $-p_2^2/4 < p_1 < 0$, an additional family of unstable periodic orbits is found, which limits on the degenerate equilibrium when $p_1 \to 0$ and on the limiting orbit on the branch of asymptotically stable periodic orbits for $p_1 \to -p_2^2/4$.

We proceed to encode a vectorized implementation of the vector field in the function `hopf`, as shown below

```
function y = hopf(x, p)

x1 = x(1,:);
x2 = x(2,:);
p1 = p(1,:);
p2 = p(2,:);

r2 = x1.^2+x2.^2;

y(1,:) = x1.*(p1+p2.*r2-r2.^2)-x2;
y(2,:) = x2.*(p1+p2.*r2-r2.^2)+x1;

end
```

In the following sequence of commands, the `ode_isol2ep` constructor is used to construct an equilibrium continuation problem.

```
>> prob = coco_prob();
>> prob = ode_isol2ep(prob, '', @hopf, [0; 0], {'p1' 'p2'}, [-1; 1]);
>> bd1  = coco(prob, 'ep_run', [], 1, 'p1', [-1 1]);
```

As predicted, a Hopf bifurcation occurs at $p_1 = 0$. We invoke the `ode_HB2po` constructor to initialize a periodic orbit continuation problem using eigenvalue and eigenvector information for the Jacobian of the vector field evaluated at the Hopf bifurcation, stored with the corresponding solution file.

```
>> HBlab = coco_bd_labs(bd1, 'HB');
>> prob = coco_prob();
>> prob = ode_HB2po(prob, '', 'ep_run', HBlab);
>> bd2  = coco(prob, 'po_run', [], 1, 'p1', [-1 1]);
```

We visualize the result of continuation by representing the family of equilibria and periodic orbits by their corresponding norms.

```
>> figure(1); clf; hold on; grid on; box on;
>> coco_plot_bd('ep_run')
>> coco_plot_bd('po_run')
>> hold off
```

As predicted, the branch of periodic orbits grows from the equilibrium in the direction of negative values of p_1 and turns around at a fold point with $p_1 \approx -1/4$.

Exercises

1. Repeat the analysis in the example for a sample of positive and negative values of p_2 and graph the solution families in a single three-dimensional diagram. Use values of the `'po.test.USTAB'` continuation parameter to distinguish between stable and unstable periodic orbits.

2. Use the `ode_SN2SN` constructor to continue along the family of saddle-node bifurcations found in the example under simultaneous variation in p_1 and p_2 and add this to the diagram created in the previous exercise. Can you explain the occurrence of a fold along the corresponding solution manifold at $p_1 = p_2 = 0$?

3. Use the `po_read_solution` utility to extract the periodic orbits found during continuation in each of the preceding exercises, and graph these together with the theoretical predictions.

4. Use the `ode_HB2po` constructor to continue families of periodic orbits from any of the Hopf bifurcations found in the examples in the `'ep'` toolbox tutorial.

---

5 A web of bifurcations – `tor`

Consider the autonomous dynamical system given by the vector field

$$F(x,p) = \begin{pmatrix} (-(\beta+\nu)x_1 + \beta x_2 - A_3 x_1^3 + B_3(x_2-x_1)^3)/r \\ \beta x_1 - (\beta+\gamma)x_2 - x_3 - B_3(x_2-x_1)^3 \\ x_2 \end{pmatrix} \tag{12}$$

in terms of the vector of state variables $x = (x_1, x_2, x_3) \in \mathbb{R}^3$ and the vector of problem parameters $p = (\nu, \beta, \gamma, r, A_3, B_3) \in \mathbb{R}^6$. For the equilibrium at the origin, a pair of conjugate eigenvalues of the Jacobian of the vector field lie on the imaginary axis when ν equals

$$-2\frac{\beta^2\gamma + r(r+\beta\gamma)(\beta+\gamma)}{\beta^2 + r(\beta+\gamma)^2 + 2\beta\gamma - \sqrt{(\beta^2 + r(\beta+\gamma)(\beta+\gamma-2))(\beta^2 + r(\beta+\gamma)(\beta+\gamma+2))}} \tag{13}$$

provided that the radical is real. Along this curve, the remaining eigenvalue equals

$$\frac{2r(\beta+\gamma)}{\beta^2 + r(\beta+\gamma)^2 + \sqrt{(\beta^2 + r(\beta+\gamma)(\beta+\gamma-2))(\beta^2 + r(\beta+\gamma)(\beta+\gamma+2))}} \tag{14}$$

and vanishes when $\beta = -\gamma$, at which point $\nu = \gamma$.

We encode the vector field and its Jacobians in the functions `tor`, `tor_dx`, and `tor_dp` as shown below.

```
function y = tor(x, p)

x1 = x(1,:);
x2 = x(2,:);
x3 = x(3,:);
nu = p(1,:);
be = p(2,:);
ga = p(3,:);
r  = p(4,:);
a3 = p(5,:);
b3 = p(6,:);

y(1,:) = ( -(be+nu).*x1 + be.*x2 - a3.*x1.^3 + b3.*(x2-x1).^3 )./r;
y(2,:) =  be.*x1 - (be+ga).*x2 - x3 - b3.*(x2-x1).^3;
y(3,:) = x2;

end


function J = tor_dx(x, p)

x1 = x(1,:);
x2 = x(2,:);
nu = p(1,:);
be = p(2,:);
ga = p(3,:);
r  = p(4,:);
```

```
a3 = p(5,:);
b3 = p(6,:);

J = zeros(3,3,numel(x1));

J(1,1,:) = (-(be+nu)-3*a3.*x1.^2-3*b3.*(x2-x1).^2)./r;
J(1,2,:) = (be+3*b3.*(x2-x1).^2)./r;
J(2,1,:) = be+3*b3.*(x2-x1).^2;
J(2,2,:) = -(be+ga)-3*b3.*(x2-x1).^2;
J(2,3,:) = -1;
J(3,2,:) = 1;

end


function J = tor_dp(x, p)

x1 = x(1,:);
x2 = x(2,:);
nu = p(1,:);
be = p(2,:);
r  = p(4,:);
a3 = p(5,:);
b3 = p(6,:);

J = zeros(3,6,numel(x1));

J(1,1,:) = -x1./r;
J(1,2,:) = (-x1+x2)./r;
J(1,4,:) = -( -(be+nu).*x1 + be.*x2 - a3.*x1.^3 + b3.*(x2-x1).^3 )./r.^2;
J(1,5,:) = -x1.^3./r;
J(1,6,:) = (x2-x1).^3./r;
J(2,2,:) = x1-x2;
J(2,3,:) = -x2;
J(2,6,:) = -(x2-x1).^3;

end
```

The following sequence of commands locates a Hopf bifurcation at $\nu \approx -0.589$ along the trivial equilibrium branch under variations in ν, in agreement with (13).

```
>> p0 = [-0.65; 0.5; -0.6; 0.6; 0.3; 0.9];
>> prob = coco_prob();
>> prob = ode_isol2ep(prob, '', @tor, [0;0;0], ...
     {'nu', 'be', 'ga', 'r', 'a3', 'b3'}, p0);
>> bd_ep = coco(prob, 'ep', [], 1, 'nu', [-0.65, -0.55]);
```

Continuation along this family of Hopf bifurcations under simultaneous variations in ν and β is then accomplished by the following sequence of commands.

```
>> lab = coco_bd_labs(bd_ep, 'HB');
>> prob = coco_prob();
>> prob = ode_HB2HB(prob, '', 'ep', lab);
>> bd_hb = coco(prob, 'hb', [], 1, {'nu', 'be'}, [-0.65, -0.55]);
```

The screen output shows the existence of a branch point along this family at $\beta = 0.6$ and $\nu = -0.6$, as predicted by the theory.

We rely on the `ode_HB2po` constructor to continue along a family of periodic orbits emanating from the initial Hopf bifurcation found above.

```
>> prob = coco_prob();
>> prob = ode_HB2po(prob, '', 'ep', lab);
>> prob = coco_set(prob, 'cont', 'NAdapt', 5, 'PtMX', [100 0]);
>> bd1  = coco(prob, 'po1', [], 1, {'nu' 'po.period'}, [-0.65, -0.55]);
```

The `'PtMX'` setting of the `'cont'` toolbox is used to restrict continuation to one direction along this family. We can continue along a secondary branch of periodic orbits emanating from the branch point found along the primary branch by using the `ode_po2po` constructor, as shown below.

```
>> BPlabs = coco_bd_labs(bd1, 'BP');
>> prob = coco_prob();
>> prob = coco_set(prob, 'cont', 'NAdapt', 5, 'PtMX', [100 0]);
>> prob = coco_set(prob, 'cont', 'branch', 'switch');
>> bd2 = coco(prob, 'po2', 'ode', 'po', 'po', 'po1', BPlabs(end), 1, ...
     {'nu' 'po.period'}, [-0.65, -0.55]);
```

Finally, we continue along a family of torus bifurcations through the point found along the secondary branch, as shown in the following sequence of commands.

```
>> TRlabs = coco_bd_labs(bd2, 'TR');
>> prob = coco_prob();
>> prob = coco_set(prob, 'cont', 'NAdapt', 5);
>> bd5 = coco(prob, 'tr1', 'ode', 'TR', 'TR', ...
     'po2', TRlabs(1), {'nu' 'po.period' 'be'}, [-0.65, -0.55]);
```

This family terminates on the branch point discovered along the curve of Hopf bifurcations of the trivial equilibrium at the origin.

Exercises

1. Use the `ode_SN2SN` constructor to continue along a family of saddle-node bifurcations that intersects the primary branch of periodic orbits under simultaneous variations in ν, β, and the orbit period.

2. Use the `ode_PD2PD` constructor to continue along a family of period-doubling bifurcations that intersects the secondary branch of periodic orbits under simultaneous variations in ν, β, and the orbit period.

3. Use the `ode_PD2po` constructor to continue along a period-doubled family of periodic orbits that emanates from the period-doubling bifurcation curve found in the previous exercise.

4. The dynamical system considered in this section is analyzed in detail in Freire, E., Rodriguez-Luis, A., Gamero, E. and Ponce, E., "A case study for homoclinic chaos in an autonomous electronic circuit: A trip from Takens-Bogdanov to Hopf-Shilnikov," *Physica D* **62**, pp. 230–253, 1993, and also in the manual for the AUTO software package. Reproduce the results in these sources using the `'ep'` and `'po'` toolbox constructors.

---

6 Homoclinic bifurcations – `marsden`

Consider the autonomous dynamical system[4] given by the vector field

$$F(x,p) = \begin{pmatrix} p_1 x_1 + x_2 + p_2 x_1^2 \\ -x_1 + p_1 x_2 + x_2 x_3 \\ (p_1^2 - 1)x_2 - x_1 - x_3 + x_1^2 \end{pmatrix} \tag{15}$$

in terms of the vector of state variables $x = (x_1, x_2, x_3) \in \mathbb{R}^3$ and the vector of problem parameters $p = (p_1, p_2) \in \mathbb{R}^2$. A vectorized encoding is implemented in the function `marsden` shown below.

```
function y = marsden(x, p)

x1 = x(1,:);
x2 = x(2,:);
x3 = x(3,:);
p1 = p(1,:);
p2 = p(2,:);

y(1,:) = p1.*x1+x2+p2.*x1.^2;
y(2,:) = -x1+p1.*x2+x2.*x3;
y(3,:) = (p1.^2-1).*x2-x1-x3+x1.^2;

end
```

For this system, the origin is an asymptotically stable equilibrium for $p_1 < 0$ and an unstable equilibrium for $p_1 > 0$. A pair of complex conjugate eigenvalues of the Jacobian of the vector field evaluated at the origin crosses the imaginary axis transversally as p_1 passes through 0.

A one-dimensional family of equilbria that passes through the Hopf bifurcation at $p_1 = 0$ is obtained using the following sequence of commands.

```
>> prob = coco_prob();
>> prob = ode_isol2ep(prob, '', @marsden, [0; 0; 0], { 'p1', 'p2' }, [-1; 6]);
>> bd1  = coco(prob, 'ep_run', [], 1, 'p1', [-1 1]);
```

[4]The Hopf bifurcation that occurs in this vector field is analyzed in Sect. 4B of Marsden, J.E. and McCracken, M., *The Hopf Bifurcation and Its Applications*, Springer-Verlag, New York, 1976.

The label of the stored solution associated with the Hopf bifurcation can be extracted using the `coco_bd_labs` utility.

```
>> HBlab = coco_bd_labs(bd1, 'HB');
```

Continuation along the family of periodic orbits emanating from the Hopf bifurcation is then achieved using the following sequence of commands.

```
>> prob = coco_prob();
>> prob = ode_HB2po(prob, '', 'ep_run', HBlab);
>> prob = coco_set(prob, 'cont', 'NAdapt', 1, 'PtMX', 50);
>> bd2  = coco(prob, 'po1', [], 1, {'p1' 'po.period'}, [-1 1]);
```

As seen in the screen output, the orbit period increases rapidly as p_1 approaches approximately -0.013, for which an equilibrium is found at $(x_1, x_2, x_3) \approx (-0.25, -0.37, 0.68)$. We visualize the time history of the longest-period orbit using the following commands.

```
>> sol = po_read_solution('', 'po1', 8);
>> plot(sol.tbp, sol.xbp(:,3))
```

The graph suggests that the periodic orbit passes close to the equilibrium and, possibly, approximates a homoclinic connecting orbit emanating from this equilibrium.

We proceed to construct an improved periodic-orbit approximation to such a homoclinic connection using the following sequence of commands.

```
>> [sol data] = coll_read_solution('po.orb', 'po1', 8);
>> f = marsden(sol.xbp', repmat(sol.p, [1 size(sol.xbp, 1)]));
>> [mn idx] = min(\\sqrt(sum(f.*f, 1)));
>> scale = 25;
>> T  = sol.T;
>> t0 = [sol.tbp(1:idx,1) ; T*(scale-1)+sol.tbp(idx+1:end,1)];
>> x0 = sol.xbp;
>> p0 = sol.p;
```

The first three commands extract the discretization point corresponding to the minimum value of the norm of the vector field along the longest-period periodic orbit found in the above run. We then insert a time segment that is a large multiple of the orbit period immediately following this discretization point. Continuation with a zero-dimensional atlas algorithm can now be used to locate a periodic orbit with period equal to its initial value, i.e., to 25 times the largest period found in the previous run.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'coll', 'NTST', data.coll.NTST);
>> prob = coco_set(prob, 'po', 'bifus', 'off');
>> prob = ode_isol2po(prob, '', @marsden, t0, x0, { 'p1' 'p2' }, p0);
>> prob = coco_set(prob, 'cont', 'NAdapt', 10);
>> prob = coco_xchg_pars(prob, 'p2', 'po.period');
>> coco(prob, 'po2', [], 0, { 'p1' 'po.orb.coll.err_TF' 'po.period' });
```

The `coco_xchg_pars` core utility is here used to assign `'p2'` to the set of active continuation parameters and `'po.period'` to the set of inactive continuation parameters, thus ensuring

that the latter is fixed during root finding. The value of 10 for the `'NAdapt'` setting of the `'cont'` toolbox implies that the trajectory discretization is changed adaptively ten times before the solution is accepted.

Continuation along a one-dimensional family of periodic orbits of very large and fixed period is now achieved using the following sequence of commands.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'po', 'bifus', 'off');
>> prob = ode_po2po(prob, '', 'po2', 2);
>> prob = coco_xchg_pars(prob, 'p2', 'po.period');
>> prob = coco_set(prob, 'cont', 'NAdapt', 1, 'PtMX', 50);
>> coco(prob, 'po3', [], 1, {'p1' 'p2' 'po.period'}, [-1 1]);
```

The collection of associated pairs of numerical values for `'p1'` and `'p2'` provides an approximate sample along a corresponding homoclinic bifurcation curve.

Exercises

1. Repeat the complete analysis for several different values of p_2.
2. Repeat the construction of an approximate homoclinic orbit for a larger value of the `scale` variable. Graph the time history of several members of the family of high-period periodic orbits by resetting the independent variable to equal 0 at the point closest to the equilibrium.
3. Use the `ode_SN2SN` constructor to continue along the family of saddle-node bifurcations detected in the example. Graph this bifurcation curve in the same diagram as the homoclinic bifurcation curve.
4. Use a combination of the `'coll'` and `'ep'` toolboxes to find a single-segment approximation to the homoclinic connecting orbit by imposing suitable boundary conditions expressed in terms of the stable and unstable eigenspaces of the corresponding equilibrium.

---

7 Canard explosions – **canard**

Consider the autonomous dynamical system given by the vector field

$$F(x,p) = \begin{pmatrix} \epsilon(a - x_2) \\ x_1 + x_2 - x_2^3 \end{pmatrix} \tag{16}$$

in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and the vector of problem parameters $p = (a, \epsilon) \in \mathbb{R}^2$. Equilibria occur at $(a^3/3 - a, a)$ and undergo Hopf bifurcations when $a = \pm 1$.

Continuation along this branch of equilibria is achieved using the following sequence of commands.

```
>> vanderpol = @(x,p) [p(2,:).*(p(1,:)-x(2,:)); x(1,:)+x(2,:)-x(2,:).^3/3];
>> pnames = { 'a' 'eps' };
>> prob = coco_prob();
>> prob = ode_isol2ep(prob, '', vanderpol, [0; 0], pnames, [0;  0.01]);
>> bd1  = coco(prob, 'ep_run', [], 1, 'a', [-1.1 1.1]);
```

We continue along the family of periodic orbits emanating from the Hopf bifurcation at $a = -1$ using the following sequence of commands.

```
>> HBlabs = coco_bd_labs(bd1, 'HB');
>> prob = coco_prob();
>> prob = coco_set(prob, 'po', 'bifus', false);
>> prob = ode_HB2po(prob, '', 'ep_run', HBlabs(1));
>> prob = coco_set(prob, 'cont', 'PtMX', 1000, 'NPR', 100);
>> prob = coco_set(prob, 'cont', 'NAdapt', 1, 'h_max', 4, 'bi_direct', false);
>> cont_args = { 1, { 'po.period' 'a' }, { [1 1000] [-1.1 0] } };
>> coco(prob, 'po_run', [], cont_args{:});
```

As seen in the screen output, the period exhibits a dramatic increase in magnitude over an extremely small interval of variation in the problem parameter a. This is known as a *canard explosion* and the periodic orbits along this near-"vertical" branch are members of a *canard family*.

Exercises

1. Repeat the construction of a canard family by continuation along the family of periodic orbits emanating from the Hopf bifurcation at $a = 1$. Explain the symmetry with the case $a = -1$.

2. Repeat the analysis for a smaller value of ϵ and compare the time histories for various members of the corresponding canard family with those obtained for $\epsilon = 0.01$. What happens when you increase the value of ϵ?

3. Use continuation to determine the dependence on ϵ of the value of a corresponding to a canard explosion and compare this to the theoretical prediction in Freire, E., Gamero, E., and Rodríguez-Luis, A.J., "First-Order Approximations for Canard Periodic Orbits in a van der Pol Electronic Oscillator," *Applied Mathematics Letters* **12**, pp. 73–78, 1999.

4. Use the `'po'` toolbox constructors to explore the occurrence of canards and period-doubling cascades of canards in the slow-fast dynamical system considered in Sekikawa, M., Inaba, N., Yoshinaga, T, and Hikihara, T., "Period-doubling cascades of canards from the extended Boenhoeffer-van der Pol oscillator," *Physics Letters A* **374(36)**, pp. 3745–3751, 2010.

8 A piecewise-smooth dynamical system – `piecewise`

Let $r = \sqrt{x_1^2 + x_2^2}$ and consider the pair of vector fields

$$F(x, p; \mathfrak{left}) = \begin{pmatrix} -x_2 + (1-r)x_1 \\ x_1 + (1-r)x_2 \end{pmatrix} \tag{17}$$

and

$$F(x, p; \mathfrak{right}) = \begin{pmatrix} \alpha(\beta - r)x_1 - (\gamma + \beta - r)x_2 \\ \alpha(\beta - r)x_2 + (\gamma + \beta - r)x_1 \end{pmatrix} \tag{18}$$

in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and vector of problem parameters $p = (\alpha, \beta, \gamma) \in \mathbb{R}^3$. Trajectory segments governed by the two vector fields may be parameterized as

$$x(t) = r(t) \begin{pmatrix} \cos\theta(t) \\ \sin\theta(t) \end{pmatrix} \tag{19}$$

in terms of the corresponding polar coordinates $r(t)$ and $\theta(t)$, satisfying the differential equations

$$\dot{r} = r(1 - r),\ \dot{\theta} = 1 \tag{20}$$

in the case of $F(x, p; \mathfrak{left})$, and

$$\dot{r} = \alpha r(\beta - r),\ \dot{\theta} = \gamma + \beta - r \tag{21}$$

in the case of $F(x, p; \mathfrak{right})$, both of which may be solved explicitly in terms of the corresponding initial values r_0 and θ_0. In particular, for positive α, the radial distance grows in time if $r_0 < 1$ for $F(x, p; \mathfrak{left})$ and $r_0 < \beta$ for $F(x, p; \mathfrak{right})$. Similarly, the radial distance decays in time if $r_0 > 1$ for $F(x, p; \mathfrak{left})$ and $r_0 > \beta$ for $F(x, p; \mathfrak{right})$. As long as $\gamma > 0$, it follows that closed curves in state space inside the annulus bounded by $r = 1$ and $r = \beta$ may be obtained by stitching together trajectory segments for each of the two vector fields.

Consider, for example, continuation along families of closed curves obtained by stitching together a trajectory segment for $F(x, p; \mathfrak{left})$ on $x_1 \leq 0$ and a trajectory segment for $F(x, p; \mathfrak{right})$ on $x_1 \geq 0$. To this end, we encode the two vector fields and their Jacobians in the functions `piecewise`, `piecewise_dx`, and `piecewise_dp`, as shown below.

```
function y = piecewise(x, p, mode)

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

r  = sqrt(x1.^2+x2.^2);

switch mode
  case 'left'
    y(1,:) = (1-r).*x1-x2;
```

```
    y(2,:) = x1+(1-r).*x2;
  case 'right'
    al = p(1,:);
    be = p(2,:);
    ga = p(3,:);

    y(1,:) = al.*(be-r).*x1-(ga+be-r).*x2;
    y(2,:) = al.*(be-r).*x2+(ga+be-r).*x1;
end

end


function J = piecewise_dx(x, p, mode)

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

r  = sqrt(x1.^2+x2.^2);
rx = x1./r;
ry = x2./r;

J  = zeros(2,2,numel(r));
switch mode
  case 'left'
    J(1,1,:) = 1-r-x1.*rx;
    J(1,2,:) = -1-x1.*ry;
    J(2,1,:) = 1-x2.*rx;
    J(2,2,:) = 1-r- x2.*ry;
  case 'right'
    al   = p(1,:);
    be   = p(2,:);
    ga   = p(3,:);

    al_x = al.*x1-x2;
    al_y = al.*x2+x1;

    J(1,1,:) = al.*be-al.*r-rx.*al_x;
    J(1,2,:) = -be-ga+r-ry.*al_x;
    J(2,1,:) = be+ga-r-rx.*al_y;
    J(2,2,:) = al.*be-al.*r-ry.*al_y;
end

end


function J = piecewise_dp(x, p, mode)

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

r  = sqrt(x1.^2+x2.^2);

J  = zeros(2,3,numel(r));
switch mode
  case 'right'
```

```
    al = p(1,:);
    be = p(2,:);

    J(1,1,:) = (be-r).*x1;
    J(1,2,:) = al.*x1-x2;
    J(1,3,:) = -x2;
    J(2,1,:) = (be-r).*x2;
    J(2,2,:) = al.*x2+x1;
    J(2,3,:) = x1;
end

end
```

The assignments

```
>> stop    = @(x,p,e) x(1);
>> stop_dx = @(x,p,e) [1 0];
>> stop_dp = @(x,p,e) zeros(1,3);
>> jump    = @(x,p,r) x;
>> jump_dx = @(x,p,r) eye(2);
>> jump_dp = @(x,p,r) zeros(2,3);
```

encode the event function $h(x, p; \mathfrak{boundary}) = x_1$ and its Jacobians, as well as the reset function $g(x, p; \mathfrak{switch}) = x$ and its Jacobians.

Continuation along the desired family under variations in β then results from the following sequence of commands

```
>> p0     = [1; 2; 1.5];
>> modes  = {'left' 'right'};
>> events = {'boundary' 'boundary'};
>> resets = {'switch' 'switch'};
>> t0 = linspace(0, pi, 100)';
>> x1 = [-sin(t0) 0.5+cos(t0)];
>> x2 = [ sin(t0) 0.5-cos(t0)];
>> t0 = {t0 0.5*t0};
>> x0 = {x1 x2};
>> prob = coco_prob();
>> prob = coco_set(prob, 'hspo.orb.bvp.seg1.coll', 'NTST', 10, 'NCOL', 6);
>> prob = coco_set(prob, 'hspo.orb.bvp.seg2.coll', 'NTST', 20, 'NCOL', 4);
>> prob = ode_isol2hspo(prob, '', {@piecewise, stop, jump}, ...
     {@piecewise_dx, stop_dx, jump_dx}, {@piecewise_dp, stop_dp, jump_dp}, ...
     modes, events, resets, t0, x0, {'al' 'be' 'ga'}, p0);
>> prob = coco_set(prob, 'cont', 'NAdapt', 5);
>> coco(prob, 'pw1', [], 1, 'be', [0 5]);
```

Here, the arguments of the `ode_isol2hspo` constructor include arrays of function handles, and arrays that identify the choice of vector field, event function, and reset function corresponding to each of the two trajectory segments. The two calls to `coco_set` assigns segment-specific values for the mesh discretization parameters, referencing the corresponding instances of the `'coll'` toolbox by their toolbox instance identifiers.

Exercises

1. Use the `ode_hspo2hspo` toolbox constructor to restart continuation under variations in α from one of the solutions found during the `'pw1'` run in the text.

2. Use the `hspo_read_solution` toolbox utility to extract trajectory information for individual solutions and confirm the prediction regarding their location within the annulus bounded by $r = 1$ and $r = \beta$.

3. The two-segment closed curves obtained during continuation correspond to periodic orbits of a piecewise-smooth dynamical system with dynamics governed by $F(x, p; \mathfrak{left})$ on $x_1 \leq 0$ and a trajectory segment for $F(x, p; \mathfrak{right})$ on $x_1 \geq 0$. Use the explicit solution to the differential equations in polar coordinates to predict the corresponding nontrivial Floquet multiplier and compare this prediction to the numerical values stored by the `'po'` toolbox in the bifurcation data cell array.

4. Perform continuation along closed curves corresponding to alternative stitching sequences with two or more trajectory segments. Repeat the stability analysis from the previous exercise and compare to the numerical predictions.

---

9 An impact oscillator – `impact`

Consider the autonomous dynamical system governed by the vector field

$$F(x,p) = \begin{pmatrix} x_2 \\ -kx_1 - cx_2 + A\cos x_3 \\ \omega \end{pmatrix} \tag{22}$$

in terms of the vector of state variables $x = (x_1, x_2, x_3) \in \mathbb{R}^2 \times \mathbb{S}$ and vector of problem parameters $p = (k, c, A, \omega, d, e) \in \mathbb{R}^6$, with state resets given by the reset map

$$g(x,p;\mathfrak{bounce}) = \begin{pmatrix} x_1 \\ -ex_2 \\ x_3 \end{pmatrix} \tag{23}$$

associated with transversal intersections with the zero-level surface of the event function

$$h(x,p;\mathfrak{impact}) = d - x_1, \tag{24}$$

and the reset map

$$g(x,p;\mathfrak{phase}) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 - 2\pi \end{pmatrix} \tag{25}$$

associated with transversal intersections with the zero-level surface of the event function

$$h(x, p; \mathfrak{phase}) = \pi - x_3. \tag{26}$$

A periodic orbit of this dynamical system is said to be *impacting* if it includes a transition governed by $g(x, p; \mathfrak{bounce})$ and to be *grazing* if it intersects the zero-level surface of $h(x, p; \mathfrak{impact})$ tangentially, i.e., at a point where $x_2 = 0$.

We may perform continuation along a family of impacting periodic orbits while monitoring for a grazing intersection. To this end, we encode the vector field and the event and reset functions in the `impact`, `impact_events`, and `impact_resets` functions shown below.

```
function y = impact(x, p, mode)

x1 = x(1,:);
x2 = x(2,:);
x3 = x(3,:);
p1 = p(1,:);
p2 = p(2,:);
p3 = p(3,:);
p4 = p(4,:);

y(1,:) = x2;
y(2,:) = p3.*cos(x3)-p2.*x2-p1.*x1;
y(3,:) = p4;

end


function y = impact_events(x, p, event)

switch event
  case 'impact'
    y = p(5,:)-x(1,:);
  case 'phase'
    y = pi-x(3,:);
end

end


function y = impact_resets(x, p, reset)

y = x;

switch reset
  case 'bounce'
    y(2,:) = -p(6,:).*y(2,:);
  case 'phase'
    y(3,:) = y(3,:)-2*pi;
end

end
```

The following sequence of commands then construct a two-segment impacting periodic-orbit continuation problem, with initial solution guess obtained by forward simulation.

```
>> p0     = [1; 0.1; 1; 1; 1; 0.8];
>> modes  = {'free' 'free'};
>> events = {'impact' 'phase'};
>> resets = {'bounce' 'phase'};
>> f      = @(t, x) impact(x, p0, 'free');
>> [t1, x1] = ode45(f, [0 3.2], [-0.98; -0.29; -pi]);
>> [t2, x2] = ode45(f, [0 3.1], [1; -1.36; 0.076]);
>> t0 = {t1 t2};
>> x0 = {x1 x2};
>> hspo_args = {{@impact, @impact_events, @impact_resets}, ...
     modes, events, resets, t0, x0, {'k' 'c' 'A' 'w' 'd' 'e'}, p0};
>> prob = coco_prob();
>> prob = coco_set(prob, 'hspo', 'bifus', false);
>> prob = ode_isol2hspo(prob, '', hspo_args{:});
```

Here, the call to `coco_set` turns off the monitoring of Floquet multipliers and bifurcation detection. This avoids the singularity in the sensitivity matrix associated with grazing contact. In order to support detection of grazing contact, we extract the second component of the initial end point of the second trajectory segment and monitor changes in its sign, as shown below.

```
>> [data, uidx] = coco_get_func_data(prob, 'hspo.orb.bvp.seg2.coll', ...
     'data', 'uidx');
>> maps = data.coll_seg.maps;
>> prob = coco_add_pars(prob, 'grazing', uidx(maps.x0_idx(2)), ...
     'graze', 'active');
>> prob = coco_add_event(prob, 'GR', 'graze', 0);
>> prob = coco_set(prob, 'cont', 'PtMX', 100, 'NAdapt', 5);
>> bd1 = coco(prob, 'impact1', [], {'A' 'graze'}, [0.01 1]);
```

As the continuation problem is implemented in terms of a constrained multi-segment boundary-value problem, the analysis produces solutions that are contained entirely in the $x_1 \leq d$ half space, as well as solutions that cross the $x_1 = d$ boundary, even though the latter violate the association of crossings of this boundary with the application of $g(x, p; \mathfrak{bounce})$.

Exercises

1. Analyze the following sequence of commands and describe the corresponding family of periodic orbits.

   ```
   >> labgr = coco_bd_labs(bd1, 'GR');
   >> prob = coco_prob();
   >> prob = coco_set(prob, 'hspo', 'bifus', false);
   >> prob = ode_hspo2hspo(prob, '', 'impact1', labgr);
   >> [data, uidx] = coco_get_func_data(prob, 'hspo.orb.bvp.seg2.coll', ...
        'data', 'uidx');
   >> maps = data.coll_seg.maps;
   >> prob = coco_add_pars(prob, 'grazing', uidx(maps.x0_idx(2)), ...
        'graze', 'active');
   >> prob = coco_xchg_pars(prob, 'graze', 'A');
   ```

```
>> prob = coco_set(prob, 'cont', 'PtMX', 100, 'NAdapt', 5);
>> coco(prob, 'impact2', [], {'w' 'A' 'graze'}, {[] [0 1]});
```

2. Use the `ode_hspo2hspo` constructor to restart continuation along several families of two-segment impacting periodic orbits from select solutions obtained during continuation in the previous exercise. Repeat this analysis for different numerical combinations of k and c.

3. Associate the identity state reset with transversal intersections with the zero-level surface of the event function $h(x, p; \mathfrak{minsep}) = x_2$, and perform continuation along a family of two-segment non-impacting periodic orbits starting from one of the solutions obtained in the first exercise.

4. Use the `ode_isol2hspo` and `ode_hspo2hspo` constructors to repeat the analysis of multi-segment periodic orbits performed in Examples 9.3 and 9.4 of *Recipes for Continuation* (cf. the `stickslip` demo).

---

10 Bang-bang excitation – **bangbang**

Consider the autonomous dynamical system governed by the nonlinear vector fields

$$F(x, p; \mathfrak{pos}) = \begin{pmatrix} x_2 \\ A - \lambda x_2 - \alpha x_1 - \epsilon x_1^3 \\ 1 \end{pmatrix} \tag{27}$$

and

$$F(x, p; \mathfrak{neg}) = \begin{pmatrix} x_2 \\ -A - \lambda x_2 - \alpha x_1 - \epsilon x_1^3 \\ 1 \end{pmatrix} \tag{28}$$

in terms of a vector of state variables $x = (x_1, x_2, x_3) \in \mathbb{R}^2 \times [0, \pi/\omega]$ and a vector of problem parameters $p = (\lambda, \alpha, \epsilon, A, \omega) \in \mathbb{R}^5$. We look for two-segment periodic orbits, where the first segment is governed by $F(x, p; \mathfrak{neg})$ and the second is governed by $F(x, p; \mathfrak{pos})$, and where transitions occur when x_3 reaches π/ω and is reset to 0.

We encode the vector fields and their Jacobians in the functions `duff`, `duff_DFDX`, and `duff_DFDP`, as shown below.

```
function y = duff(x, p, mode)

x1 = x(1,:);
x2 = x(2,:);
la = p(1,:);
al = p(2,:);
ep = p(3,:);
```

```
A  = p(4,:);
om = p(5,:);

switch mode
  case 'neg'
    A=-A;
end

y(1,:) = x2;
y(2,:) = A-la.*x2-al.*x1-ep.*x1.^3;
y(3,:) = ones(1,numel(om));

end


function J = duff_DFDX(x, p, mode)

x1 = x(1,:);
la = p(1,:);
al = p(2,:);
ep = p(3,:);

J = zeros(3,3,numel(x1));
J(1,2,:) = 1;
J(2,1,:) = -al-3*ep.*x1.^2;
J(2,2,:) = -la;

end


function J = duff_DFDP(x, p, mode)

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

J = zeros(3,5,numel(x1));
J(2,1,:) = -x2;
J(2,2,:) = -x1;
J(2,3,:) = -x1.^3;
J(2,4,:) = 1;
switch mode
  case 'neg'
    J(2,4,:) = -1;
end

end
```

We construct an initial solution guess, allowing the transient dependence on initial conditions to die out, by repeated calls to `ode45` as shown in the following sequence of commands.

```
>> w0 = 1.1;
>> p0 = [0.2; 1; 1; 26; w0];
>> x0 = [0; 0; 0];
>> for i=1:10
     [~, y0] = ode45(@(t,x) duff(x,p0,'neg'), [0 pi/w0], x0);
     x0 = y0(end,:)';
```

```
    [~, y1] = ode45(@(t,x) duff(x,p0,'pos'), [0 pi/w0], x0);
    x0 = y1(end,:)';
  end
>> x0 = [x0(1:2);0];
>> [t1, x1] = ode45(@(t,x) duff(x,p0,'neg'), [0 pi/w0], x0);
>> x0 =  [x1(end,1:2) 0]';
>> [t2, x2] = ode45(@(t,x) duff(x,p0,'pos'), [0 pi/w0], x0);
```

We rely on the `ode_isol2hspo` constructor to build the corresponding continuation problem, as shown below.

```
>> modes  = {'neg' 'pos'};
>> events = {'phase' 'phase'};
>> resets = {'phase' 'phase'};
>> t0     = {t1 t2};
>> x0     = {x1 x2};
>> duff_events    = @(x,p,~) pi/p(5)-x(3);
>> duff_events_dx = @(x,p,~) [0 0 -1];
>> duff_events_dp = @(x,p,~) [0 0 0 0 -pi/p(5)^2];
>> duff_resets    = @(x,p,~) [x(1);x(2);0];
>> duff_resets_dx = @(x,p,~) [1 0 0; 0 1 0; 0 0 0];
>> duff_resets_dp = @(x,p,~) zeros(3,5);
>> prob = coco_prob();
>> prob = coco_set(prob, 'coll', 'NTST', 200);
>> prob = ode_isol2hspo(prob, '', {@duff, duff_events, duff_resets}, ...
     {@duff_DFDX, duff_events_dx, duff_resets_dx}, ...
     {@duff_DFDP, duff_events_dp, duff_resets_dp}, ...
     modes, events, resets, t0, x0, {'la' 'al' 'eps' 'A' 'om'}, p0);
```

Continuation then proceeds according to the settings for the `'cont'` toolbox assigned by a call to `coco_set`.

```
>> prob = coco_set(prob, 'cont', 'h', 2, 'h_max', 20, 'PtMX', 200, 'NAdapt', 1);
>> coco(prob, 'run1', [], 1, 'om', [0.8 1.5]);
```

Several branch points and saddle-node bifurcations found during continuation may be used to restart continuation along secondary branches of (further constrained) two-segment periodic orbits.

Exercises

1. Use the `ode_hspo2hspo` constructor with the `'branch'` property of the `'cont'` settings equal to `'switch'` to restart continuation along a secondary branch emanating from one of the branch points found in the original run.

2. Use the `ode_SN2SN` constructor to restart continuation along a family of saddle-node bifurcations of two-segment periodic orbits emanating from one of the saddle-node bifurcations found in the original run.

3. Use the `ode_PD2PD` constructor to restart continuation along a family of period-doubling bifurcations of two-segment periodic orbits emanating from the period-doubling bifurcations found in the first exercise.

4. Use the `ode_PD2hspo` constructor to restart continuation along a family of four-segment periodic orbits emanating from one of the period-doubling bifurcations found in the first exercise.

---

11 Optimization – `int_optim`

Consider the problem of finding stationary points of the functional

$$x(t) \mapsto \int_0^T \frac{x_1(t)}{1+x_2^2(t)}\,\mathrm{d}t \tag{29}$$

evaluated on a family of period-T solutions of the autonomous dynamical system[5]

$$\dot{x}_1 = \frac{1}{p_1}\left(-p_4\left(\frac{x_1^3}{3}-x_1\right)+\frac{x_3-x_1}{p_2}-x_2\right),\ \dot{x}_2 = x_1-p_3,\ \dot{x}_3 = -\frac{x_3-x_1}{p_2} \tag{30}$$

emanating from a Hopf bifurcation along the equilibrium point family

$$x_1 = p_3,\ x_2 = \frac{p_3 p_4}{3}(3-p_3^2),\ x_3 = p_3. \tag{31}$$

We implement the equilibrium continuation problem in COCO using the following call to the `ode_isol2ep` constructor:

```
>> funcs  = {@mvdP, @mvdP_dx, @mvdP_dp, @mvdP_dxdx, @mvdP_dxdp, @mvdP_dpdp};
>> x0     = [0;0;0];
>> pnames = {'p1', 'p2', 'p3', 'p4'};
>> p0     = [0.5; 4; 0; 2];
>> prob = coco_prob();
>> prob = ode_isol2ep(prob, '', funcs{:}, x0, pnames, p0);
```

where the encodings in `@mdvP`, `@mvdP_dx`, `@mvdP_dp`, `@mvdP_dxdx`, `@mvdP_dxdp`, and `@mvdP_dpdp` are given below.

```
function y = mvdP(x, p)

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

[5]This modified Van-der-Pol model of a nonlinear electronic circuit is analyzed in Sect. 2.5 of Rodríguez Luis, *Bifurcaciones Multiparamétricas en Osciladores Autónomos*, PhD dissertation, Universidad de Sevilla, 1991.

```
p1 = p(1,:);
p2 = p(2,:);
p3 = p(3,:);
p4 = p(4,:);

y(1,:) = (-p4.*(x1.^3/3-x1) + (x3-x1)./p2 - x2)./p1;
y(2,:) = x1-p3;
y(3,:) = -(x3-x1)./p2;

end


function J = mvdP_dx(x, p)

x1 = x(1,:);
p1 = p(1,:);
p2 = p(2,:);
p4 = p(4,:);

J = zeros(3,3,numel(x1));
J(1,1,:) = (-p4.*(x1.^2-1)-1./p2)./p1;
J(2,1,:) = 1;
J(3,1,:) = 1./p2;
J(1,2,:) = -1./p1;
J(1,3,:) = 1./p1./p2;
J(3,3,:) = -1./p2;

end


function J = mvdP_dp(x, p)

x1 = x(1,:);
x2 = x(2,:);
x3 = x(3,:);
p1 = p(1,:);
p2 = p(2,:);
p4 = p(4,:);

J = zeros(3,4,numel(x1));
J(1,1,:) = (p4.*(x1.^3/3-x1) - (x3-x1)./p2 + x2)./p1.^2;
J(1,2,:) = (x1-x3)./p1./p2.^2;
J(3,2,:) = (x3-x1)./p2.^2;
J(2,3,:) = -1;
J(1,4,:) = (x1-x1.^3/3)./p1;

end


function dJ = mvdP_dxdx(x, p)

x1 = x(1,:);
p1 = p(1,:);
p4 = p(4,:);

dJ = zeros(3,3,3,numel(1));
```

```
dJ(1,1,1,:) = -2*p4.*x1./p1;

end


function dJ = mvdP_dxdp(x, p)

x1 = x(1,:);
p1 = p(1,:);
p2 = p(2,:);
p4 = p(4,:);

dJ = zeros(3,3,4,numel(x1));
dJ(1,1,1,:) = (p4*(x1.^2-1)+1./p2)./p1.^2;
dJ(1,1,2,:) = 1./p1./p2.^2;
dJ(3,1,2,:) = -1./p2.^2;
dJ(1,1,4,:) = (1-x1.^2)./p1;
dJ(1,2,1,:) = 1./p1.^2;
dJ(1,3,1,:) = -1./p1.^2./p2;
dJ(1,3,2,:) = -1./p1./p2.^2;
dJ(3,3,2,:) = 1./p2.^2;

end


function dJ = mvdP_dpdp(x, p)

x1 = x(1,:);
x2 = x(2,:);
x3 = x(3,:);
p1 = p(1,:);
p2 = p(2,:);
p4 = p(4,:);

dJ = zeros(3,4,4,numel(x1));
dJ(1,1,1,:) = 2*(-p4.*(x1.^3/3-x1) + (x3-x1)./p2 - x2)./p1.^3;
dJ(1,2,1,:) = (x3-x1)./p1.^2./p2.^2;
dJ(1,4,1,:) = (x1.^3/3-x1)./p1.^2;
dJ(1,2,2,:) = 2*(x3-x1)./p1./p2.^3;
dJ(3,2,2,:) = 2*(x1-x3)./p2.^3;
dJ(1,1,2,:) = (x3-x1)./p1.^2./p2.^2;
dJ(1,1,4,:) = (x1.^3/3-x1)./p1.^2;

end
```

We trace a one-dimensional solution manifold under variations in p_3 using the `coco` entry-point function, as shown below.

```
>> bd1 = coco(prob, 'ep_run', [], 1, 'p3', [0 1]);
```

From the screen output, we note the detection of a Hopf bifurcation at $p_3 \approx 0.93756$. The `ode_HB2po` constructor can be used to build a periodic-orbit continuation problem for the family of orbits emanating from this bifurcation point, as shown in the next sequence of commands.

```
>> HBlab = coco_bd_labs(bd1, 'HB');
>> prob = coco_prob();
>> prob = coco_set(prob, 'coll', 'NTST', 20);
>> prob = coco_set(prob, 'cont', 'NAdapt', 1, 'PtMX', [0 100]);
>> prob = coco_set(prob, 'po', 'bifus', false);
>> prob = ode_HB2po(prob, '', 'ep_run', HBlab);
```

Here, we use an initial discretization with 20 mesh intervals and allow for an adaptive remeshing after each complete step of continuation. We restrict computation to 100 complete steps along one of the directions on the corresponding solution manifold, and turn off bifurcation detection by setting the `'bifus'` option of the `'po'` toolbox to `false`.

We proceed to append a monitor function corresponding to an arbitrary integral functional of the form

$$x(t) \mapsto \int_0^T g(x(t))\,\mathrm{d}t \tag{32}$$

using the sequence of commands shown below.

```
>>[fdata, uidx] = coco_get_func_data(prob, 'po.orb.coll', 'data', 'uidx');
>> data = int_init_data(fdata, 'po.orb');
>> maps = data.coll_seg.maps;
>> uidx = uidx([maps.xbp_idx; maps.T_idx]);
>> prob = coco_add_func(prob, 'po.orb.int', @int, @int_du, data, ...
     'inactive', 'po.orb.int', 'uidx', uidx, 'remesh', @int_remesh);
```

In the encoding of `int`, we rely on the trajectory discretization stored in the function data structure for the corresponding `'coll'` trajectory segment to approximate the integral functional by quadrature on the collocation nodes.

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

pr   = data.pr;
maps = pr.coll_seg.maps;
mesh = pr.coll_seg.mesh;

T    = u(pr.T_idx);
x    = u(pr.xbp_idx);

xcn = reshape(maps.W*x, maps.x_shp);
gcn = pr.ghan(xcn);
gcn = mesh.gka.*gcn;

y = (0.5*T/maps.NTST)*mesh.gwt*gcn';

end
```

The corresponding Jacobian is encoded in `int_du` as shown below.

```
function [data, J] = int_du(prob, data, u)

pr   = data.pr;
maps = pr.coll_seg.maps;
mesh = pr.coll_seg.mesh;
```

```
T = u(pr.T_idx);
x = u(pr.xbp_idx);

xcn = reshape(maps.W*x, maps.x_shp);
gcn = pr.ghan(xcn);
gcn = mesh.gka.*gcn;

gdxcn = pr.ghan_dx(xcn);
gdxcn = mesh.gdxka.*gdxcn;
gdxcn = sparse(maps.gdxrows, maps.gdxcols, gdxcn(:));

J_xbp = (0.5*T/maps.NTST)*mesh.gwt*gdxcn*maps.W;
J_T   = (0.5/maps.NTST)*mesh.gwt*gcn';

J = [J_xbp J_T];

end
```

We initialize the associated function data structure using the call to the `int_init_data` function whose encoding is shown below.

```
function data = int_init_data(fdata, oid)

data.coll_seg = fdata.coll_seg;
data.xbp_idx  = data.coll_seg.maps.xbp_idx;
data.T_idx    = data.xbp_idx(end) + 1;
data.ghan     = @ghan;
data.ghan_dx  = @ghan_dx;
data.oid      = oid;

data = coco_func_data(data);

end
```

Encodings of `ghan` and `ghan_dx` corresponding to the specific integrand of interest are shown below.

```
function y = ghan(x)

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

y  = x1./(1+x2.^2);

end


function J = ghan_dx(x)

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

J  = zeros(1,3,numel(x1));
J(1,1,:) = 1./(1+x2.^2);
```

```
J(1,2,:) = -2*x1.*x2./(1+x2.^2).^2;

end
```

Finally, in recognition of the adaptive updates to the trajectory discretization, we include the `'remesh'` flag in the call to `coco_add_func`, followed by a function handle to the following encoding that makes corresponding updates to the `'po.orb.int'` function data structure and function dependency index set.

```
function [prob, status, xtr] = int_remesh(prob, data, chart, old_u, old_V)

cid = coco_get_id(data.oid, 'coll');
[fdata, uidx] = coco_get_func_data(prob, cid, 'data', 'uidx');
data = int_init_data(fdata, data.oid);
maps = data.coll_seg.maps;
uidx = uidx([maps.xbp_idx; maps.T_idx]);

xtr    = [];
prob   = coco_change_func(prob, data, 'uidx', uidx);
status = 'success';

end
```

Before proceeding to consider an implementation of the corresponding adjoint problem, let $\hat{x}(\tau) = x(t(\tau))$ be obtained from the transformation

$$t(\tau) = T \int_0^\tau \kappa(s)\,\mathrm{d}s, \tag{33}$$

in terms of a given positive function $\kappa(s)$ that satisfies the condition

$$\int_0^1 \kappa(s)\,\mathrm{d}s = 1. \tag{34}$$

It follows that

$$\dot{\hat{x}}_1 = \frac{T\kappa}{p_1}\left(-p_4\left(\frac{\hat{x}_1^3}{3} - \hat{x}_1\right) + \frac{\hat{x}_3 - \hat{x}_1}{p_2} - \hat{x}_2\right),\ \dot{\hat{x}}_2 = T\kappa\left(\hat{x}_1 - p_3\right)\ \dot{\hat{x}}_3 = -T\kappa\frac{\hat{x}_3 - \hat{x}_1}{p_2} \tag{35}$$

while the functional now takes the form

$$\hat{x}(t) \mapsto T\int_0^1 \kappa(\tau)\frac{\hat{x}_1(\tau)}{1+\hat{x}_2^2(\tau)}\,\mathrm{d}\tau. \tag{36}$$

Next, given some function $\hat{x}_0(\tau)$, consider the Lagrangian

$$
\begin{aligned}
L\left(\hat{x}(\tau), p, T, \mu_p, \mu_{\mathrm{int}}, \ell_{\mathrm{ode}}(\tau), \ell_{\mathrm{bc}}, \ell_{\mathrm{phase}}, \eta_p, \eta_{\mathrm{int}}\right) &= \mu_{\mathrm{int}} \\
&+ \int_0^1 \ell_{\mathrm{ode},1}(\tau)\Bigg(\dot{\hat{x}}_1(\tau) - \frac{T\kappa(\tau)}{p_1}\Bigg(- p_4\left(\frac{\hat{x}_1^3(\tau)}{3} - \hat{x}_1(\tau)\right) + \\
&\qquad\qquad \frac{\hat{x}_3(\tau) - \hat{x}_1(\tau)}{p_2} - \hat{x}_2(\tau)\Bigg)\Bigg)\,\mathrm{d}\tau \\
&+ \int_0^1 \ell_{\mathrm{ode},2}(\tau)\left(\dot{\hat{x}}_2(\tau) - T\kappa(\tau)\left(\hat{x}_1(\tau) - p_3\right)\right)\,\mathrm{d}\tau \\
&+ \int_0^1 \ell_{\mathrm{ode},3}(\tau)\left(\dot{\hat{x}}_3(\tau) + T\kappa(\tau)\frac{\hat{x}_3(\tau) - \hat{x}_1(\tau)}{p_2}\right)\,\mathrm{d}\tau \\
&+ \ell_{\mathrm{bc}}^{\top}\left(\hat{x}(1) - \hat{x}(0)\right) + \ell_{\mathrm{phase}}\int_0^1 \dot{\hat{x}}_0(\tau)^{\top}\hat{x}(\tau)\,\mathrm{d}\tau \\
&+ \eta_{\mathrm{int}}\left(T\int_0^1 \kappa(\tau)\frac{\hat{x}_1(\tau)}{1+\hat{x}_2^2(\tau)}\,\mathrm{d}\tau - \mu_{\mathrm{int}}\right) + \eta_p^{\top}\left(p - \mu_p\right)
\end{aligned}
\tag{37}
$$

in terms of the continuation parameters μ_p and μ_{int}, and the Lagrange multipliers $\ell_{\mathrm{ode}}(\tau)$, ℓ_{bc}, ℓ_{phase}, η_p, and η_{int}. Necessary conditions for stationary points along the constraint manifold correspond to points $(\hat{x}(\tau), p, T, \mu_p, \mu_{\mathrm{int}}, \ell_{\mathrm{ode}}(\tau), \ell_{\mathrm{bc}}, \ell_{\mathrm{phase}}, \eta_p, \eta_{\mathrm{int}})$ for which $\delta L = 0$ for any infinitesimal variations $\delta\hat{x}(\tau)$, δp, δT, $\delta\mu_p$, $\delta\mu_{\mathrm{int}}$, $\delta\ell_{\mathrm{ode}}(\tau)$, $\delta\ell_{\mathrm{bc}}$,$\delta\ell_{\mathrm{phase}}$, $\delta\eta_p$, $\delta\eta_{\mathrm{int}}$. For example, at a stationary point, variations with respect to ℓ_{phase} yield the integral phase condition

$$
\int_0^1 \dot{\hat{x}}_0(\tau)^{\top}\hat{x}(\tau)\,\mathrm{d}\tau = 0, \tag{38}
$$

while variations with respect to μ_{int} and μ_p show that $\eta_{\mathrm{int}} = 1$ and $\eta_p = 0$.

Variations with respect to $\hat{x}(\tau)$, p, and T yield adjoint conditions that are linear in the Lagrange multipliers. Terms associated with the components of $\ell_{\mathrm{ode}}(\tau)$ and ℓ_{bc}, as well as the scalar ℓ_{phase}, are appended without difficulty to the continuation problem by invoking the `adjt_isol2po` constructor, as shown below.

```
>> prob = adjt_isol2po(prob, '');
```

In the absence of a pre-packaged constructor, we append the corresponding terms associated with the integral functional using the following sequence of commands:

```
>> [fdata, axidx] = coco_get_adjt_data(prob, 'po.orb.coll', 'data', 'axidx');
>> data = adjt_int_init_data(fdata, 'po.orb');
>> opt  = data.coll_opt;
>> aidx = axidx([opt.xcn_idx; opt.T_idx]);
>> prob = coco_add_adjt(prob, 'po.orb.int', @adjt_int, @adjt_int_du, data, ...
       'd.po.orb.int', 'aidx', aidx, 'remesh', @adjt_int_remesh);
```

Here, the encodings of `adjt_int` and `adjt_int_du`, shown below, again rely on the discretization of the corresponding trajectory segment.

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

pr   = data.pr;
maps = pr.coll_seg.maps;
mesh = pr.coll_seg.mesh;

T = u(pr.T_idx);
x = u(pr.xbp_idx);

xcn = reshape(maps.W*x, maps.x_shp);
gcn = pr.ghan(xcn);
gcn = mesh.gka.*gcn;

gdxcn = pr.ghan_dx(xcn);
gdxcn = mesh.gdxka.*gdxcn;

J_xbp = (0.5*T/maps.NTST)*gdxcn(:)';
J_T   = (0.5/maps.NTST)*mesh.gwt*gcn';

y = [J_xbp J_T];

end


function [data, J] = adjt_int_du(prob, data, u)

pr   = data.pr;
maps = pr.coll_seg.maps;
mesh = pr.coll_seg.mesh;
opt  = pr.int_opt;

T = u(pr.T_idx);
x = u(pr.xbp_idx);

xcn = reshape(maps.W*x, maps.x_shp);

gdxdxcn = pr.ghan_dxdx(xcn);
gdxdxcn = mesh.gdxdxka.*gdxdxcn;
gdxdxcn = sparse(opt.gdxdxrows1, opt.gdxdxcols1, gdxdxcn(:))*maps.W;
J       = (0.5*T/maps.NTST)*sparse(opt.gdxdxrows2, opt.gdxdxcols2, ...
  gdxdxcn(opt.gdxdxidx), opt.dJrows, opt.dJcols);

gdxcn   = pr.ghan_dx(xcn);
gdxcn   = mesh.gdxka.*gdxcn;
J       = J + (0.5/maps.NTST)*sparse(opt.gdxdTrows, opt.gdxdTcols, ...
  gdxcn(:), opt.dJrows, opt.dJcols);

gdxcn   = mesh.gwt*sparse(maps.gdxrows, maps.gdxcols, gdxcn(:))*maps.W;
J       = J + (0.5/maps.NTST)*sparse(opt.gdTdxrows, opt.gdTdxcols, ...
  gdxcn(:), opt.dJrows, opt.dJcols);

end
```

The function data structure associated with `adjt_int` and `adjt_int_du` is initialized in the encoding of `adjt_int_init_data`, shown below.

```
function data = adjt_int_init_data(fdata, oid)

data.coll_seg  = fdata.coll_seg;
data.coll_opt  = fdata.coll_opt;
data.xbp_idx   = data.coll_seg.maps.xbp_idx;
data.T_idx     = data.xbp_idx(end) + 1;
data.ghan      = @ghan;
data.ghan_dx   = @ghan_dx;
data.ghan_dxdx = @ghan_dxdx;

seg  = fdata.coll_seg;
maps = seg.maps;
int  = seg.int;

NCOL = int.NCOL;
NTST = maps.NTST;
xdim = int.dim;

cndim  = NCOL*xdim;
bpdim  = xdim*(NCOL+1);
xbpdim = NTST*(NCOL+1)*xdim;
xcnnum = NTST*NCOL;
xcndim = NTST*NCOL*xdim;
addim  = xcndim+1;

% Derivative of (T/2N)*gxcn with respect to xbp:
rows = 1 + xcnnum*repmat(0:xdim-1, [xdim 1]);
rows = repmat(rows(:), [1 xcnnum]) + repmat(0:xcnnum-1, [xdim^2 1]);
opt.gdxdxrows1 = rows;
cols = 1:xcndim;
opt.gdxdxcols1 = repmat(reshape(cols, [xdim xcnnum]), [xdim 1]);

opt.gdxdxrows2 = ones(cndim*xbpdim, 1);
cols = 1 + xdim*(0:NCOL-1);
cols = repmat(cols(:), [1 xdim]) + repmat(0:xdim-1, [NCOL 1]);
cols = repmat(cols(:), [1 bpdim]) + addim*repmat(0:bpdim-1, [cndim 1]);
cols = repmat(cols(:), [1 NTST]) + ...
  (cndim+addim*bpdim)*repmat(0:NTST-1, [cndim*bpdim 1]);
opt.gdxdxcols2 = cols;

idx = 1:NCOL;
idx = repmat(idx(:), [1 xdim]) + xcnnum*repmat(0:xdim-1, [NCOL 1]);
idx = repmat(idx(:), [1 bpdim]) + xcndim*repmat(0:bpdim-1, [cndim 1]);
idx = repmat(idx(:), [1 NTST]) + ...
  (NCOL+cndim*xbpdim)*repmat(0:NTST-1, [cndim*bpdim 1]);
opt.gdxdxidx = idx;

% Derivative of (T/2N)*gxcn with respect to T:
opt.gdxdTrows = ones(xcndim,1);
opt.gdxdTcols = addim*xbpdim  + (1:xcndim)';

% Derivative of (1/2N)*w*g' with respect to xbp:
opt.gdTdxrows = ones(xbpdim,1);
opt.gdTdxcols = xcndim + 1 + addim*(0:xbpdim-1)';
```

```
opt.dJrows = 1;
opt.dJcols = addim*(xbpdim+1);

data.int_opt = opt;
data.oid = oid;

data = coco_func_data(data);

end
```

Finally, we encode adaptive updates to the function data structure and the `'aidx'` index vector in the function `adjt_int_remesh`, shown below.

```
function [prob, status, xtr, ftr] = adjt_int_remesh(prob, data, chart, lb, Vlb)

cid = coco_get_id(data.oid, 'coll');
[fdata, axidx] = coco_get_adjt_data(prob, cid, 'data', 'axidx');
data = adjt_int_init_data(fdata, data.oid);
opt  = data.coll_opt;
aidx = axidx([opt.xcn_idx; opt.T_idx]);

xtr    = [];
ftr    = 1;
prob   = coco_change_adjt(prob, data, 'aidx', aidx, 'l0', lb, 'vecs', Vlb);
status = 'success';

end
```

Continuation then proceeds along the one-dimensional family of periodic orbits according to the following commands:

```
>> cont_args = {1, {'po.orb.int', 'p3', 'd.p1', 'd.p2', 'd.p4', ...
     'd.po.orb.int'}};
>> bd2  = coco(prob, 'po_run', [], cont_args{:});
```

Here, the entries in the definition of `cont_args` indicate that μ_{int}, $\mu_{p,3}$, $\eta_{p,1}$, $\eta_{p,2}$, $\eta_{p,4}$, and η_{int} are allowed to vary, while $\mu_{p,1}$, $\mu_{p,2}$, $\mu_{p,4}$, and $\eta_{p,3}$ remain fixed.

By linearity, and given the default zero initialization of the Lagrange multipliers, the Lagrange multipliers remain equal to 0 throughout this initial continuation run. By construction, a local extremum in the value of μ_{int} along the one-dimensional solution family corresponds to a branch point of the corresponding restricted continuation problem. The following sequence of commands reconstructs the periodic orbit continuation problem and the associated contributions to the adjoint equations, and initializes the associated vector of continuation variables and candidate tangent vector for continuation along the secondary branch through this point.

```
>> BPlab = coco_bd_labs(bd2, 'BP');
>> prob = coco_prob();
>> prob = coco_set(prob, 'cont', 'NAdapt', 1);
>> prob = coco_set(prob, 'po', 'bifus', false);
>> prob = ode_po2po(prob, '', 'po_run', BPlab);
>> prob = adjt_po2po(prob, '', 'po_run', BPlab);
```

We append the integral monitor function to the continuation problem according to the following sequence of commands.

```
>> [fdata, uidx] = coco_get_func_data(prob, 'po.orb.coll', 'data', 'uidx');
>> data = int_init_data(fdata, 'po.orb');
>> maps = data.coll_seg.maps;
>> uidx = uidx([maps.xbp_idx; maps.T_idx]);
>> prob = coco_add_func(prob, 'po.orb.int', @int, @int_du, data, ...
     'inactive', 'po.orb.int', 'uidx', uidx, 'remesh', @int_remesh);
```

These are identical to those used in the initial construction, since no new continuation variables are associated with the monitor function.

In contrast, the reconstruction and initialization of the associated contributions to the adjoint equations require that we extract solution data for the branch point stored to disk during the `'po_run'` continuation run. In the commands below, the call to `coco_read_adjoint` stores the value of the Lagrange multiplier η_{int} in `chart.x`.

```
>> chart = coco_read_adjoint('po.orb.int', 'po_run', BPlab, 'chart');
```

We use this information in the call below to `coco_add_adjt`.

```
>> [fdata, axidx] = coco_get_adjt_data(prob, 'po.orb.coll', 'data', 'axidx');
>> data = adjt_int_init_data(fdata, 'po.orb');
>> opt  = data.coll_opt;
>> aidx = axidx([opt.xcn_idx; opt.T_idx]);
>> prob = coco_add_adjt(prob, 'po.orb.int', @adjt_int, @adjt_int_du, data, ...
     'd.po.orb.int', 'aidx', aidx, 'remesh', @adjt_int_remesh, ...
     'l0', chart.x, 'tl0', chart.t);
```

Continuation then proceeds along the secondary branch of periodic orbits according to the following commands:

```
>> prob = coco_set(prob, 'cont', 'branch', 'switch');
>> cont_args = {1, {'d.po.orb.int', 'po.orb.int', 'p3', 'd.p1', 'd.p2', ...
     'd.p4'}, [0 1]};
>> bd3 = coco(prob, 'po_run_lagrange1', [], cont_args{:});
```

We terminate continuation when `'d.po.orb.int'` equals 1, consistent with the requirement that $\eta_{\text{int}} = 1$ at a stationary point.

In order to impose the further condition that $\eta_{p,2} = 0$, we apply a third stage of continuation along a one-dimensional solution manifold characterized by fixed values of p_1, p_4, $\eta_{p,3}$, and $\eta_{\text{int}} = 1$, with varying values of p_2, p_3, $\eta_{p,1}$, $\eta_{p,2}$, and $\eta_{p,4}$. To this end, we use the solution data stored for the second `'EP'` point in the previous continuation run to construct the periodic orbit continuation problem and the associated contributions to the adjoint equations, as shown below.

```
>> EPlab = coco_bd_labs(bd3, 'EP');
>> prob = coco_prob();
>> prob = coco_set(prob, 'cont', 'NAdapt', 1);
>> prob = coco_set(prob, 'po', 'bifus', false);
>> prob = ode_po2po(prob, '', 'po_run_lagrange1', EPlab(2));
```

```
>> prob = adjt_po2po(prob, '', 'po_run_lagrange1', EPlab(2));
```

Similarly, we append the integral monitor function using the identical sequence of commands to those used in the previous stages of continuation.

```
>> [fdata, uidx] = coco_get_func_data(prob, 'po.orb.coll', 'data', 'uidx');
>> data = int_init_data(fdata, 'po.orb');
>> maps = data.coll_seg.maps;
>> uidx = uidx([maps.xbp_idx; maps.T_idx]);
>> prob = coco_add_func(prob, 'po.orb.int', @int, @int_du, data, ...
     'inactive', 'po.orb.int', 'uidx', uidx, 'remesh', @int_remesh);
```

The construction of the contributions to the adjoint equations associated with the integral monitor function shown below omits reference to a candidate tangent vector, since there is no ambiguity in this choice at a regular solution point.

```
>> chart = coco_read_adjoint('po.orb.int', 'po_run_lagrange1', EPlab(2), ...
     'chart');
>> [fdata, axidx] = coco_get_adjt_data(prob, 'po.orb.coll', 'data', 'axidx');
>> data = adjt_int_init_data(fdata, 'po.orb');
>> opt  = data.coll_opt;
>> aidx = axidx([opt.xcn_idx; opt.T_idx]);
>> prob = coco_add_adjt(prob, 'po.orb.int', @adjt_int, @adjt_int_du, data, ...
     'd.po.orb.int', 'aidx', aidx, 'remesh', @adjt_int_remesh, ...
     'l0', chart.x);
```

Finally, in order to detect and terminate at a zero crossing of $\eta_{p,2}$, we use the `coco_add_event` utility as shown below.

```
>> prob = coco_add_event(prob, 'OPT', 'BP', 'd.p2', '==', 0);
>> cont_args = {1, {'d.p2', 'po.orb.int', 'p3', 'd.p1', 'p2', 'd.p4'}};
>> bd4  = coco(prob, 'po_run_lagrange2', [], cont_args{:});
```

Each `'OPT'` point then corresponds to a stationary point of the integral functional under simultaneous variations in p_2 and p_3.

Exercises

1. Starting from an `'OPT'` point located during the previous continuation run, use an additional stage of continuation to impose the further condition that $\eta_{p,1} = 0$.

2. Derive the adjoint equations for the Lagrangian L and explain the encoding of the integral monitor function, its adjoint, and their derivatives in `int`, `int_du`, `adjt_int`, and `adjt_int_du`.

3. Modify the encodings associated with the integral monitor function and the corresponding contributions to the adjoint equations to enable optimization of an integral

functional of the form

$$\int_0^T g(x(t), p, T)\,\mathrm{d}t,$$

including the case that $g(x(t), p, T) = p_3/T$.

4. Repeat the analysis in the `ops` demo of the manual to the AUTO continuation package[6] using the observation and code from the previous exercise, or directly using the `'po'` toolbox without any further construction.

---

12 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.

12.1 Zero problems

Recall the dependence of the `'coll'` trajectory segment zero functions on the column matrix υ_{bp} of state variables on the mesh of base points, the initial time T_0, the interval length T, and the problem parameters. For continuation of single-segment periodic orbits, the zero problem is given in terms of the vector of continuation variables $u = (\upsilon_{bp}, T_0, T, p)$ by appending the zero function $\Phi(u)$ to the `'coll'` toolbox trajectory segment zero problem, where

$$\Phi : (\upsilon_{bp}, T_0, T, p) \mapsto \begin{pmatrix} \upsilon_i - \upsilon_f \\ n^T \cdot \upsilon_{bp} \end{pmatrix} \tag{39}$$

in the case of an autonomous dynamical systems and

$$\Phi : (\upsilon_{bp}, T, p) \mapsto \upsilon_i - \upsilon_f \tag{40}$$

in the case of a non-autonomous dynamical system. Here, υ_i and υ_f denote the state vectors for the initial and final end points of the corresponding trajectory segment. In the autonomous case, the continuation variable T_0 is constrained by an initially inactive continuation parameter encoded by the corresponding `'coll'` instance. In the non-autonomous case, the `'po'` instance encodes two initially inactive continuation parameters corresponding the continuation variables T_0 and T. In either case, the dimensional deficit equals q, where q is the number of problem parameters.

In the autonomous case, the phase condition $n^T \cdot \upsilon_{bp} = 0$ identifies a unique member of the family of periodic orbits obtained by arbitrary shifts in time. The vector n is updated

[6] Doedel, E.J. and Oldeman, B.E. AUTO-07P: Continuation and Bifurcation Software. 2012.

before each continuation step by applying a linear transformation to a previously obtained v_{bp}. This linear transformation depends on the discretization encoded in the `'coll'` zero problem, and changes in response to adaptive remeshing of the trajectory discretization.

In the current implementation of the `'po'` toolbox, the zero problem for continuation of multi-segment periodic orbits applies only to autonomous dynamical systems. Let M denote the number of trajectory segments. To the collection of `'coll'` trajectory segment zero problems, the `'po'` toolbox then appends a family $\Phi(v_{1,i}, \ldots, v_{M,i}, v_{1,f}, \ldots, v_{M,f}, p)$ of event and reset conditions that associate each segment with a unique point of termination on a zero-level surface of an event function, such that the terminal point is mapped to the initial point on the subsequent segment by some reset function. Specifically, suppose that the vector field, event function, and reset function associated with the j-th trajectory segment are given by

$$(x, p) \mapsto f(x, p; \mathfrak{m}_j),\ (x, p) \mapsto h(x, p; \mathfrak{e}_j), \text{ and } (x, p) \mapsto g(x, p; \mathfrak{r}_j), \tag{41}$$

respectively, in terms of the mode label $\mathfrak{m}_j$, event label $\mathfrak{e}_j$, and reset label $\mathfrak{r}_j$. Then,

$$\Phi : \left(\{v_{j,i}\}_{j=1}^{M}, \{v_{j,f}\}_{j=1}^{M}, p\right) \mapsto \begin{pmatrix} h(v_{1,f}, p; \mathfrak{e}_1) \\ v_{2,i} - g(v_{1,f}, p; \mathfrak{r}_1) \\ h(v_{2,f}, p; \mathfrak{e}_2) \\ v_{3,i} - g(v_{2,f}, p; \mathfrak{r}_2) \\ \vdots \\ h(v_{M,f}, p; \mathfrak{e}_M) \\ v_{1,i} - g(v_{M,f}, p; \mathfrak{r}_M) \end{pmatrix}. \tag{42}$$

Collectively, the sequences $\{\mathfrak{m}_j\}_{j=1}^{M}$, $\{\mathfrak{e}_j\}_{j=1}^{M}$, and $\{\mathfrak{r}_j\}_{j=1}^{M}$ are referred to as the orbit signature. The dimensional deficit of the multi-segment periodic orbit continuation problem equals q.

In the current implementation of the `'po'` toolbox, continuation of bifurcations of single- or multi-segment periodic orbits relies on the simultaneous continuation of the corresponding trajectory segments together with solutions to the associated variational equations. Recall from the `'coll'` tutorial the notation Δ_{bp} for the unknown values of a solution to the variational equation along some trajectory segment on the corresponding mesh of base points. Then, in the case of continuation of saddle-node and period-doubling bifurcations, Δ_{bp} consists of single column, whereas it has two columns in the case of continuation of torus bifurcations.

For single-segment periodic orbits in an autonomous dynamical system, the zero problem for continuation of saddle-node bifurcations is now obtained by appending the zero function $\Phi(v_i, \Delta_i, \Delta_f, p, b)$ to the periodic orbit continuation problem. Here, the i and f subscripts again reference the initial and final end points along the trajectory segment, and

$$\Phi(v_i, \Delta_i, \Delta_f, p, b) \mapsto \begin{pmatrix} \Delta_i + bF(v_i, p) - \Delta_f \\ F(v_i, p)^T \cdot \Delta_i \\ \Delta_i^T \cdot \Delta_i - 1 \end{pmatrix}. \tag{43}$$

The vector $F(\upsilon_i, p)$ is an eigenvector of the monodromy matrix corresponding to the eigenvalue 1. When $b \neq 0$, this formulation ensures that Δ_i is a generalized eigenvector corresponding to the same eigenvalue. The corresponding dimensional deficit equals $q - 1$.

For single-segment periodic orbits in a non-autonomous dynamical system, the zero problem for continuation of saddle-node bifurcations is obtained by appending the zero function $\Phi(\Delta_i, \Delta_f)$ to the periodic orbit zero problem, where

$$\Phi(\Delta_i, \Delta_f) \mapsto \begin{pmatrix} \Delta_i - \Delta_f \\ \Delta_i^T \cdot \Delta_i - 1 \end{pmatrix}. \tag{44}$$

The corresponding dimensional deficit equals q.

For single-segment periodic orbits, the zero problem for continuation of period-doubling bifurcations is obtained by appending the zero function $\Phi(\Delta_i, \Delta_f)$ to the periodic orbit zero problem, where

$$\Phi(\Delta_i, \Delta_f) \mapsto \begin{pmatrix} \Delta_i + \Delta_f \\ \Delta_i^T \cdot \Delta_i - 1 \end{pmatrix} \tag{45}$$

whether the dynamical system is autonomous or non-autonomous. The corresponding dimensional deficit equals $q - 1$ in the autonomous case and q in the non-autonomous case.

For single-segment periodic orbits, the zero problem for continuation of torus bifurcations is obtained by appending the zero function $\Phi(\Delta_i, \Delta_f, a, b)$ to the periodic orbit zero problem, where

$$\Phi(\Delta_i, \Delta_f, a, b) \mapsto \begin{pmatrix} \Delta_{f,1} - a\Delta_{i,1} + b\Delta_{i,2} \\ \Delta_{f,2} - a\Delta_{i,2} - b\Delta_{i,1} \\ \Delta_{i,1}^T \cdot \Delta_{i,1} + \Delta_{i,2}^T \cdot \Delta_{i,2} - 1 \\ \Delta_{i,1}^T \cdot \Delta_{i,2} \\ a^2 + b^2 - 1 \end{pmatrix} \tag{46}$$

whether the dynamical system is autonomous or non-autonomous. Here, the second subscript denotes the corresponding column of the solution to the trajectory segment variational problem. The corresponding dimensional deficit equals $q - 1$ in the autonomous case and q in the non-autonomous case.

For multi-segment periodic orbits, denote by

$$P_j = g_{,x}(\upsilon_{j,f}, p; \mathfrak{r}_j) \cdot \left(I_{n_j} - \frac{f(\upsilon_{j,f}, p; \mathfrak{m}_j) \cdot h_{,x}(\upsilon_{j,f}, p; \mathfrak{e}_j)}{h_{,x}(\upsilon_{j,f}, p; \mathfrak{e}_j) \cdot f(\upsilon_{j,f}, p; \mathfrak{m}_j)} \right) \tag{47}$$

the projection matrix that maps the solution of the variational equation associated with the final end point along the j-th trajectory segment to an initial condition for the variational equation along the subsequent trajectory segment. Then, the zero problem for continuation of saddle-node bifurcations of multi-segment periodic orbits is obtained by appending the zero function

$$\Phi : \left(\{\upsilon_{j,f}\}_{j=1}^M, \{\Delta_{j,i}\}_{j=1}^M, \{\Delta_{j,f}\}_{j=1}^M, p \right) \mapsto \begin{pmatrix} P_1 \cdot \Delta_{1,f} - \Delta_{2,i} \\ P_2 \cdot \Delta_{2,f} - \Delta_{3,i} \\ \cdots \\ P_M \cdot \Delta_{M,f} - \Delta_{1,i} \\ \Delta_{1,i}^T \cdot \Delta_{1,i} - 1 \end{pmatrix} \tag{48}$$

to the multi-segment periodic orbit zero problem. Similarly, the zero problem for continuation of period-doubling bifurcations of multi-segment periodic orbits is obtained by appending the zero function

$$\Phi : \left(\{\upsilon_{j,f}\}_{j=1}^{M}, \{\Delta_{j,i}\}_{j=1}^{M}, \{\Delta_{j,f}\}_{j=1}^{M}, p\right) \mapsto \begin{pmatrix} P_1 \cdot \Delta_{1,f} - \Delta_{2,i} \\ P_2 \cdot \Delta_{2,f} - \Delta_{3,i} \\ \dots \\ P_M \cdot \Delta_{M,f} + \Delta_{1,i} \\ \Delta_{1,i}^T \cdot \Delta_{1,i} - 1 \end{pmatrix} \tag{49}$$

to the multi-segment periodic orbit zero problem. Finally, the zero problem for continuation of torus bifurcations of multi-segment periodic orbits is obtained by appending the zero function

$$\Phi : \left(\{\upsilon_{j,f}\}_{j=1}^{M}, \{\Delta_{j,i}\}_{j=1}^{M}, \{\Delta_{j,f}\}_{j=1}^{M}, p, a, b\right) \mapsto \begin{pmatrix} P_1 \cdot \Delta_{1,f,1} - \Delta_{2,i,1} \\ P_1 \cdot \Delta_{1,f,2} - \Delta_{2,i,2} \\ P_2 \cdot \Delta_{2,f,1} - \Delta_{3,i,1} \\ P_2 \cdot \Delta_{2,f,2} - \Delta_{3,i,2} \\ \dots \\ P_{M-1} \cdot \Delta_{M-1,f,1} - \Delta_{M,i,1} \\ P_{M-1} \cdot \Delta_{M-1,f,2} - \Delta_{M,i,2} \\ P_M \cdot \Delta_{M,f,1} - a\Delta_{1,i,1} + b\Delta_{1,i,2} \\ P_M \cdot \Delta_{M,f,2} - a\Delta_{1,i,2} - b\Delta_{1,i,1} \\ \Delta_{1,i,1}^T \cdot \Delta_{1,i,1} + \Delta_{1,i,2}^T \cdot \Delta_{1,i,2} - 1 \\ \Delta_{1,i,1}^T \cdot \Delta_{1,i,2} \\ a^2 + b^2 - 1 \end{pmatrix} \tag{50}$$

to the multi-segment periodic orbit zero problem. As before, the last subscript refers to the corresponding column of the solution to the variational equations.

12.2 Calling syntax

The calling syntax for generic `'po'` toolbox constructors 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_isol2po` toolbox constructor, the `varargin` input argument equals

```
varargin = coll [opts]
```

where `coll` equals the `varargin` argument of the `ode_isol2coll` constructor in the `'coll'` toolbox, i.e.,

```
coll = fcns t0 x0  [pnames]  p0 [opts]
```

where

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

in the case of an autonomous vector field and

```
fcns = @f [@dfdx [@dfdp [@dfdt [@dfdxdx [@dfdxdp [@dfdpdp
                              [@dfdtdx [@dfdtdp [@dfdtdt]]]]]]]]]
```

for an non-autonomous vector field. Here, `@f` denotes a required function handle to the encoding of the operator F, and each of the optional arguments `@dfdx`, `@dfdp`, `@dfdt`, `@dfdxdx`, `@dfdxdp`, `@dfdpdp`, `@dfdtdx`, `@dfdtdp`, and `@dfdtdt` is either an empty array (`[]`) or a function handle to the corresponding array of partial derivatives with respect to the state variables, the problem parameters, or time, respectively. An initial solution guess for the time mesh, the state variables, and the problem parameters is given by the `t0`, `x0`, and `p0` input arguments, respectively. Notably, if adjoint equations are to be constructed using the `adjt_isol2po` constructor, then the preceding call to `ode_isol2po` must include explicit function handles to encodings of the Jacobians with respect to x, p, and (as appropriate) t, respectively.

In the case of the `ode_isol2hspo` constructor, the `varargin` input argument adheres to the syntax

```
varargin = { funcs sig {t0...} {x0...} [pnames] p0 [opts] }
```

where

```
funcs    = fun [ fun_dx [ fun_dp [ fun_dxdx [ fun_dxdp [ fun_dpdp ] ] ] ] ]
fun      = { @f @e @r }
fun_dx   = { (@dfdx | '[]') (@dedx | '[]') (@drdx | '[]') }
fun_dp   = { (@dfdp | '[]') (@dedp | '[]') (@drdp | '[]') }
fun_dxdx = { (@dfdxdx | '[]') (@dedxdx | '[]') (@drdxdx | '[]') }
fun_dxdp = { (@dfdxdp | '[]') (@dedxdp | '[]') (@drdxdp | '[]') }
fun_dpdp = { (@dfdpdp | '[]') (@dedpdp | '[]') (@drdpdp | '[]') }
sig      = { mode... } { event... } { reset... }
```

and where the ellipsis indicates a sequence of arguments of the same type of length equal to the number of segments. Here, `@f`, `@e`, and `@r` denote required function handles to encodings of the vector field $f(x, p, \mathfrak{m})$, the event function $h(x, p; \mathfrak{e})$, and the reset function $g(x, p; \mathfrak{r})$. Each of the optional arguments `@dfdx`, `@dedx`, `@drdx`, `@dfdp`, `@dedp`, `@drdp`, `@dfdxdx`, `@dedxdx`, `@drdxdx`, `@dfdxdp`, `@dedxdp`, `@drdxdp`, `@dfdpdp`, `@dedpdp`, and `@drdpdp` 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_isol2hspo` constructor, then the preceding call to `ode_isol2hspo` must include explicit function handles to encodings of the Jacobians with respect to x and p, respectively.

The `sig` argument contains the orbit signature, encoded in three cell arrays representing the collection of mode, event, and reset labels, respectively. An initial solution guess for the time mesh and state variables for each segment is provided by the elements of the `{t0...}` and `{x0...}` arguments, respectively. The corresponding values of the problem parameters

are given by p0. 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.

For each of the remaining toolbox constructors, the varargin input argument adheres to the syntax

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

In all cases, run denotes a string identifying a previous run and lab is a numeral identifying the corresponding solution file. The optional argument soid denotes a source object instance identifier, in the case that this differs from oid.

For toolbox constructors used to build single-segment periodic orbit continuation problems, the optional opts argument may equal either of the strings '-po-end' or '-end-po'. Similarly, for toolbox constructors used to build multi-segment periodic orbit continuation problems, the optional opts argument may equal either of the strings '-hspo-end' or '-end-hspo'. In either case, this denotes explicitly the end of the sequence of arguments to a 'po' toolbox constructor. For ode_po2po and ode_hspo2hspo, 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 the ode_po2po, ode_HB2po, and ode_PD2po constructors, opts may also contain the string '-var' followed by a numerical matrix, indicating the simultaneous continuation of solutions to the corresponding variational problem. In this case, each column of the matrix corresponds to a perturbation to the initial point on the trajectory segment. Similarly, in the case of ode_hspo2hspo and ode_PD2hspo constructors, opts may contain the string '-var' followed by a cell array of numerical matrices, again indicating the simultaneous continuation of solutions to each of the corresponding variational problems. Each element of the cell array represents a collection of perturbations to the initial point on the corresponding trajectory segment.

12.3 Adjoint functions

For continuation of general single-segment periodic orbits, the contributions to the adjoint equations associated with variations in υ_{bp}, T_0, T, and p are expressed in terms of the Jacobians $\partial_t F(t,x,p)$, $\partial_x F(t,x,p)$, and $\partial_p F(t,x,p)$ 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_isol2po constructor, following a preceding call to the ode_isol2po constructor that includes function handles to explicit encodings of these Jacobians, or to the ode_HB2po constructor that includes reference to a Hopf bifurcation detected during a preceding stage of equilibrium point continuation. Specifically, in the call

```
prob = adjt_isol2po(prob, oid)
```

the oid argument denotes an object identifier associated with the toolbox instance created by the preceding call to ode_isol2po or ode_HB2po. The corresponding components of λ are

initialized to 0.

If the preceding call to `ode_isol2po` includes an explicit list of parameter labels, then the corresponding additions to the adjoint equations are automatically encoded by the call to `adjt_isol2po`. The corresponding components of the vector of continuation multipliers η are initialized to 0.

In a similar fashion, a call to `ode_po2po` may be followed by a call to `adjt_po2po` 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.

For continuation of multi-segment periodic orbits, the contributions to the adjoint equations associated with variations in v_{bp}, T_0, T, and p are additionally expressed in terms of the Jacobians $\partial_x f(x,p)$, $\partial_x g(x,p)$, $\partial_x h(x,p)$, $\partial_p f(x,p)$, $\partial_p g(x,p)$, and $\partial_p h(x,p)$, 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_isol2hspo` constructor, following a preceding call to the `ode_isol2hspo` constructor that includes function handles to explicit encodings of these Jacobians. Specifically, in the call

```
prob = adjt_isol2hspo(prob, oid)
```

the `oid` argument denotes an object identifier associated with the toolbox instance created by the preceding call to `ode_isol2hspo`. The corresponding components of λ are initialized to 0.

If the preceding call to `ode_isol2hspo` includes an explicit list of parameter labels, then the corresponding additions to the adjoint equations are automatically encoded by the call to `adjt_isol2hspo`. The corresponding components of the vector of continuation multipliers η are initialized to 0.

In a similar fashion, a call to `ode_hspo2hspo` may be followed by a call to `adjt_hspo2hspo` 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.

12.4 Continuation parameters

The inclusion of the `pnames` optional argument in the call to either of the `ode_isol2po` or `ode_isol2hspo` toolbox constructors ensures that the continuation problem structure encodes embedded continuation parameters that are 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. Notably, `ode_HB2po` reuses information about such embedded continuation parameters stored to disk at a Hopf bifurcation during a previous run with the `'ep'` toolbox. A subsequent call to `adjt_isol2po`, `adjt_po2po`, `adjt_isol2hspo`, or

adjt_hspo2hspo ensures the encoding in the continuation problem structure of an accompanying set of initially inactive embedded continuation parameters, which correspond to an associated subset of the vector of continuation multipliers η (initialized to 0), and with labels obtained by appending 'd.' to the original string labels.

For single-segment periodic orbit continuation problems, the 'po' constructors encode an embedded monitor function whose output equals the interval length T with identifier 'OID.po.period' in terms of the 'po' object instance identifier OID (the period is omitted when OID equals the empty string). The corresponding continuation parameter is initially active in the case of an autonomous dynamical system and initially inactive otherwise. In the case of a non-autonomous vector field, the 'po' constructors encode an additional embedded monitor function whose output equals the initial time T_0 with identifier 'OID.po.tinit'. The corresponding continuation parameter is initially inactive. For multi-segment periodic orbit continuation problems, the 'po' constructors encode an embedded monitor function whose output equals the total interval length $\sum_j T_j$ with identifier 'OID.hspo.period'. The corresponding continuation parameter is initially active.

If the 'bifus' option of a 'po' instance is set to true (as it is by default), the ode_isol2po, ode_po2po, ode_HB2po, and ode_PD2po constructors also encode the four nonembedded continuation parameters 'OID.po.test.SN', 'OID.po.test.PD', 'OID.po.test.TR', and 'OID.po.test.USTAB', associated with detection of saddle-node bifurcations, period-doubling bifurcations, torus bifurcation and neutral saddle points, and with monitoring the Lyapunov stability (the number of unstable Floquet multipliers) of the periodic orbit, respectively. In this case, changes to the sign of the first three of these continuation parameters trigger the detection of special points denoted by 'SN', 'PD', and 'TR', respectively. If the 'NSA' option of the 'po' instance is set to true (contrary to its default value of false), then neutral saddles, denoted by 'NSA', are also located.

If the 'bifus' option of an 'hspo' instance is set to true (as it is by default), the ode_isol2hspo, ode_hspo2hspo, and ode_PD2hspo constructors also encode the four nonembedded continuation parameters 'OID.hspo.test.SN', 'OID.hspo.test.PD', 'OID.hspo.test.TR', and 'OID.hspo.test.USTAB', associated with detection of saddle-node bifurcations, period-doubling bifurcations, torus bifurcation and neutral saddle points, and with monitoring the Lyapunov stability (the number of unstable Floquet multipliers) of the multi-segment periodic orbit, respectively. In this case, changes to the sign of the first three of these continuation parameters trigger the detection of special points denoted by 'SN', 'PD', and 'TR', respectively. If the 'NSA' option of the 'hspo' instance is set to true (contrary to its default value of false), then neutral saddles, denoted by 'NSA', are also located.

12.5 Toolbox settings

Optional settings associated with embedded instances of the 'coll' toolbox may be assigned non-default values using the coco_set utility. These include the initial number of discretization intervals ('NTST' with default value 10), the degree of the interpolating polynomials ('NCOL' with default value 4), and the discretization error tolerance used to trigger the 'MXCL' special point and (by default) to define the adaptation window for adaptive atlas

algorithms. As discussed in the 'coll' tutorial, it may be best to only change the discretization error tolerance by changing the global COCO tolerance in order to ensure consistency with the value used by the nonlinear solver.

For a multi-segment periodic orbit continuation problem, each trajectory segment is associated with a separate instance of the 'coll' settings. The coco_set core utility can be used to set these individually or collectively, as described in *Recipes for Continuation* and the 'coll' tutorial.

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

```
prob = coco_set(prob, 'OID.po', ...
```

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

```
prob = coco_set(prob, 'PID.po', ...
```

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

```
prob = coco_set(prob, 'po', ...
```

An analogous pattern applies to 'hspo' instances. As explained in *Recipes for Continuation*, precedence is given to settings defined using the most specific path identifier. See the output of the po_settings and hspo_settings utilities for a list of supported settings and their default or current values.

12.6 Toolbox output

By definition, the bifurcation data cell array stored during continuation of single-segment periodic orbits and returned by the coco entry-point function (given a receiving variable) includes columns inherited from the embedded 'coll' instance with object instance identifier OID.po.orb, as well as four columns with headers '||OID.x||_{2,MPD}', 'MAX(OID.x)', 'MIN(OID.x)', and 'OID.eigs' with data given by a quadrature-approximation of the $\mathcal{L}_2[0,1]$ norm of the deviation of the time-rescaled periodic orbit from its state-space mean, the maximum and minimum entries of each state variable along the orbit, and the corresponding Floquet multipliers, respectively, and with OID representing the 'po' object instance identifier. For multi-segment periodic orbits, the bifurcation data cell array includes columns inherited from the embedded 'coll' instances with object instance identifiers OID.hspo.orb.bvp.seg1, OID.hspo.orb.bvp.seg2, and so on, as well as a column with header 'OID.eigs' with data given by the corresponding Floquet multipliers. 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 single-segment periodic orbits, the sol output argument of the po_read_solution utility contains

- the time instances corresponding to the mesh of base points (in the `tbp` field),
- the values of the state variables on the mesh of base points (in the `xbp` field),
- the interval length (in the `T` field),
- the vector of problem parameters (in the `p` field).

Similarly, for multi-segment periodic orbits, the `sol` output argument of the toolbox solution extractor `hspo_read_solution` contains

- a cell array with one entry per segment containing the time instances corresponding to the mesh of base points (in the `tbp` field),
- a cell array with one entry per segment containing the values of the state variables on the mesh of base points (in the `xbp` field),
- a cell array with one entry per segment containing the interval length (in the `T` field),
- the vector of problem parameters (in the `p` field).

For saddle-node bifurcations of single-segment periodic orbits in autonomous dynamical systems, the `sol` output argument contains

- the unit (generalized) eigenvector v of the monodromy matrix corresponding to the eigenvalue 1 (in the `var.v` field),
- the coefficient of the vector field f at the end point of the trajectory segment along the unit projection of v onto the nullspace of the monodromy matrix (in the `var.b` field).

For saddle-node bifurcations of single-segment periodic orbits in non-autonomous dynamical systems, the `var.v` field contains the unit eigenvector of the monodromy matrix corresponding to the eigenvalue 1. Similarly, for period-doubling bifurcation points of single-segment periodic orbits, the field `var.v` contains the unit eigenvector of the monodromy matrix corresponding to the eigenvalue -1, whether the dynamical system is autonomous or non-autonomous. Finally, for torus bifurcation points, the `sol` output argument contains

- the real and imaginary parts of a complex eigenvector of the monodromy matrix corresponding to an eigenvalue with magnitude 1 (in the `var.v` field, normalized to be mutually orthogonal with unit length for the real part), and
- cosine and sine of the phase of the eigenvalue (in the `var.a` and `var.b` fields).

For bifurcations of multi-segment periodic orbits, the `var.v` field of the `sol` output argument contains a cell array whose first entry corresponds to the content described in the single-segment case, and whose subsequent entries are the forward images of perturbations to the initial point on the first trajectory segment for each subsequent segment.

The `po_plot_theme` and `hspo_plot_theme` toolbox utilities define the default visualization theme for the `'po'` toolbox. For example, the command

```
>> thm = po_plot_theme('po')
```

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

```
>> thm_SN = po_plot_theme('po.SN')
>> thm_PD = po_plot_theme('po.PD')
>> thm_TR = po_plot_theme('po.TR')
```

assign the default themes for continuation of saddle-node, period-doubling, and torus bifurcation points, respectively, to the variables `thm_SN`, `thm_PD`, and `thm_TR`. Notably, when visualizing the results of continuation of general periodic orbits, the continuation parameter `'OID.po.test.USTAB'` is used to distinguish branches of stable and unstable orbits, respectively. In this case, to include markers identifying saddle-node bifurcations, period-doubling bifurcations, neutral saddles, or torus bifurcations, the labels `'SN'`, `'PD'`, `'NSA'`, or `'TR'` should be added to the `special` field of the problem-specific plotting theme. The same principles apply to continuation of multi-segment periodic orbits with instances of `'hspo'`.

12.7 Developer's interface

Continuation problems constructed with the `'po'` toolbox constructors may be embedded in larger continuation problem that contain additional continuation variables, zero functions, and/or monitor functions. Each `'po'` instance is associated with a toolbox instance identifier obtained by prepending an object instance identifier to the string `'po'`. Similarly, each `'hspo'` instance is associated with a toolbox instance identifier obtained by prepending an object instance identifier to the string `'hspo'`. The object instance identifier of the instance of `'coll'` embedded in an instance of `'po'` is obtained by appending `'orb'` to the `'po'` toolbox instance identifier. The object instance identifier of the 1st (2nd, 3rd, ...) instance of `'coll'` embedded in an instance of `'hspo'` is obtained by appending `'orb.bvp.seg1'` (`'orb.bvp.seg2'`, `'orb.bvp.seg3'`, ...) to the `'hspo'` toolbox instance identifier.

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 single- or multi-segment periodic orbit continuation problem. In particular, the `cid` field of the data structure of a `'po'` instance contains the toolbox instance identifier for the corresponding embedded `'coll'` instance. Similarly, the `bvid` field of the data structure of an `'hspo'` instance contains the toolbox instance identifier for the corresponding embedded `'bvp'` instance. These may be used to extract function data associated with embedded `'coll'` instances and to assign `'coll'` toolbox settings for individual trajectory segments.

The toolbox data structure associated with single- or multi-segment periodic orbit continuation problems contains a number of implementation-dependent internal fields, whose use may change in the future. Accessing such internal fields is deprecated.

12.8 Deprecated constructors

Beginning with the January 2025 release of COCO and the `'po'` toolbox, use of the `ode_BP2po`, `ode_BP2hspo`, `adjt_BP2po`, and `adjt_BP2hspo` constructors is deprecated. Instead, use the `ode_po2po`, `ode_hspo2hspo`, `adjt_po2po`, and `adjt_hspo2hspo` constructors and set the `'branch'` property of the `'cont'` settings to `'switch'`.