The Trajectory Collocation Toolbox

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

Frank Schilder
Department of Mathematics
Technical University of Denmark

March 22, 2020

Contents

1 Introduction

The 'coll' toolbox is a basic toolbox for continuation along families of constrained collections of trajectory segments 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 'coll' toolbox belongs to the 'ode' toolbox family, and is modeled on the 'coll', 'bvp', 'msbvp', and 'varcoll' toolboxes, described in *Recipes for Continuation*[1].

The 'coll' toolbox supports adaptive discretization of the state-space representation of each trajectory segment in terms of a continuous piecewise-polynomial function of t, and of the governing differential equations in terms of derivative conditions at a collection of collocation nodes associated with the Gauss-Legendre quadrature nodes. Notably, the discretizations for different trajectory segments in a continuation problem are independent.

The 'coll' toolbox 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 'coll' toolbox supports simultaneous continuation of trajectory segments and solutions to the corresponding variational equations that lie in $\mathbb{R}^{n \times m}$ for arbitrary m. Such solutions describe the linear sensitivity to perturbations and correspond to directional derivatives of the flow operator corresponding to the dynamical system.

The 'coll' 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.

For continuation of individual trajectory segments or mutually constrained collections of trajectory segments, the 'coll' toolbox supports the construction of the associated adjoint equations[2,3].

The toolbox user interface is defined by the coll_read_solution and bvp_read_solution utilities, which read solution and toolbox data from disk, and by the toolbox constructors

- ode_isol2coll for continuation along a family of individual trajectory segments from an initial solution guess;
- ode_coll2coll for continuation along a family of individual trajectory segments 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_BP2coll` for continuation along a family of individual trajectory segments from a branch point along a secondary branch;
- `ode_isol2bvp` for continuation along a family of mutually constrained collections of trajectory segments from an initial solution guess;
- `ode_bvp2bvp` for continuation along a family of mutually constrained collections of trajectory segments from a saved solution point;
- `ode_BP2bvp` for continuation along a family of mutually constrained collections of trajectory segments from a branch point along a secondary branch.

The additional constructors `adjt_isol2coll`, `adjt_coll2coll`, and `adjt_BP2coll` contribute terms to the adjoint equations associated with the zero and monitor functions appended to a continuation problem by the `ode_isol2coll`, `ode_coll2coll`, and `ode_BP2coll` constructors, respectively. The additional constructors `adjt_isol2bvp`, `adjt_bvp2bvp`, and `adjt_BP2bvp` contribute terms to the adjoint equations associated with the zero and monitor functions appended to a continuation problem by the `ode_isol2bvp`, `ode_bvp2bvp`, and `ode_BP2bvp` constructors, respectively.

Usage is illustrated in the following several examples. Each example corresponds to fully documented code in the `coco/coll/examples` folder in the COCO release. Slight differences between the code included below and the example implementations in `coco/coll/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 shooting method – `catenary`

Consider the autonomous two-point boundary-value problem

$$\dot{x}_1 = x_2,\ \dot{x}_2 = (1 + x_2^2)/x_1,\ x_1(0) = 1,\ x_1(1) = p \tag{2}$$

in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and the scalar problem parameter $p \in \mathbb{R}$. Solutions correspond to stationary curves $s \mapsto f(s)$, and their derivatives, for the integral functional

$$\int_0^1 f(s)\sqrt{1 + f'(s)^2}\,\mathrm{d}s \tag{3}$$

in the space of functions that satisfy the boundary conditions $f(0) = 1$ and $f(1) = p$.

For arbitrary initial conditions $x_1(0)$ and $x_2(0)$, solutions to the associated initial-value problem are given by

$$x_1(t) = \frac{x_1(0)}{\sqrt{1+x_2^2(0)}} \cosh\left(\frac{\sqrt{1+x_2^2(0)}}{x_1(0)} t + \operatorname{arcsinh} x_2(0)\right) \tag{4}$$

and

$$x_2(t) = \sinh\left(\frac{\sqrt{1+x_2^2(0)}}{x_1(0)} t + \operatorname{arcsinh} x_2(0)\right). \tag{5}$$

For each p, the boundary-value problem then corresponds to a solution of the nonlinear equation

$$\frac{1}{\sqrt{1+x_2^2(0)}} \cosh\left(\sqrt{1+x_2^2(0)} + \operatorname{arcsinh} x_2(0)\right) = p \tag{6}$$

Since the left-hand side is convex with a unique global minimum at $x_2(0) \approx -2.26$, it follows that there are no solutions to the boundary-value problem for $p \lesssim 0.587$ and two solutions for $p \gtrsim 0.587$. As an example, when $p = \cosh 1$, the boundary-value problem is satisfied by the functions $x_1(t) = \cosh t$ and $x_2 = \sinh t$, for which $x_2(0) = 0$.

We construct a family of solutions to the boundary-value problem for admissible values of p by first constructing a family of trajectory segments that satisfy the boundary condition at $t = 0$, but are defined only on the interval $[0, T]$ for varying interval length T. To this end, we encode the vector field in the anonymous function `cat`, as shown in the following command

```
>> cat = @(x,p) [x(2,:); (1+x(2,:).^2)./x(1,:)];
```

The encoding is vectorized and autonomous. A corresponding trajectory segment is given by the single-point time history assigned below to the `t0` and `x0` variables.

```
>> t0 = 0;
>> x0 = [1 0];
```

Here, `t0` encodes a one-dimensional array of time instances and `x0` encodes a two-dimensional array of the corresponding points in state space, with one row per time instant.

We compute a family of trajectory segments under variations in T by invoking the `coco` entry-point function as shown in the sequence of commands below.

```
>> prob = coco_prob();
>> prob = ode_isol2coll(prob, '', cat, t0, x0, []);
>> data = coco_get_func_data(prob, 'coll', 'data');
>> maps = data.coll_seg.maps;
>> prob = coco_add_pars(prob, 'pars', ...
     [maps.x0_idx; maps.x1_idx(1); maps.T_idx], ...
     {'y1s' 'y2s' 'y1e' 'T'});
>> prob = coco_set(prob, 'cont', 'NAdapt', 10);
>> cont_args = {1, {'T' 'y1e' 'coll.err' 'coll.err_TF'}, [0 1]};
>> bd = coco(prob, 'coll1', [], cont_args{:});
```

Here, the coco_prob core utility assigns an empty continuation problem structure to prob. The ode_isol2coll constructor appends an instant of the collocation zero problem to the continuation problem structure. The second input argument to this constructor is used to create a unique toolbox instance identifier, in this case the default value 'coll'. The empty bracket in the last input argument corresponds to the absence of any problem parameters in the encoding of the vector field. The dimensional deficit of the continuation problem encoded thus far in prob equals the number of state-space dimensions plus 1, i.e., 3.

The coco_get_func_data utility is used in the next line to extract the function data structure associated with the 'coll' toolbox instance. The data.coll_seg.maps field contains construction-independent integer indices for distinct elements in the vector of continuation variables that can be used to refer to the variables representing the initial and final points on the trajectory segment, as well as to the interval length. The coco_add_pars utility appends four monitor functions and corresponding continuation parameters to the continuation problem. In particular, the two elements of the vector of continuation variables that correspond to the initial point on the trajectory segment are constrained to the continuation parameters 'y1s' and 'y2s'. Similarly, the element of the vector of continuation variables corresponding to the first component of the final point on the trajectory segment is constrained to the continuation parameter 'y1e'. Finally, the element of the vector of continuation variables corresponding to the interval length is constrained to the continuation parameter 'T'. By default, these continuation parameters are all inactive. The dimensional deficit of the continuation problem encoded thus far in prob therefore equals $3 - 4 = -1$.

The call to the coco entry-point function identifies the run by the string identifier 'coll1'. It uses the empty bracket to indicate the complete encoding of the corresponding extended continuation problem and the initial assignment of inactive continuation parameters in prob. It identifies the desired dimension of the solution manifold by the integer 1 in the cont_args input argument. To accommodate this dimensionality, the 'T' and 'y1e' continuation parameters are released and 'T' is allowed to vary across the interval $[0, 1]$. The screen output also includes a discretization error estimate coll.err, as well as the ratio coll.err_TF between this error estimate and a default tolerance. Here, the coco_set core utility is used to assign the non-default value 10 to the 'NAdapt' setting of the 'cont' toolbox family, in order to indicate that the discretization should be adaptively remeshed every ten continuation steps.

The final trajectory segment obtained during continuation satisfies the boundary-value problem for some value of p, and provides a starting point for continuation in p. The sequence of commands below extracts the corresponding solution label from the bd output of the previous run, and uses this to reconstruct a continuation problem structure that is identical to the previous one.

```
>> labs = coco_bd_labs(bd, 'EP');
>> prob = coco_prob();
>> prob = ode_coll2coll(prob, '', 'coll1', labs(end));
>> data = coco_get_func_data(prob, 'coll', 'data');
>> maps = data.coll_seg.maps;
>> prob = coco_add_pars(prob, 'pars', ...
```

```
    [maps.x0_idx; maps.x1_idx(1); maps.T_idx], ...
    {'y1s' 'y2s' 'y1e' 'T'});
>> prob = coco_set(prob, 'cont', 'NAdapt', 10, 'PtMX', 200);
>> cont_args = {1, {'y1e' 'y2s' 'coll.err' 'coll.err_TF'}, [0 3]};
>> coco(prob, 'coll2', [], cont_args{:});
```

Here, the third and fourth input arguments to the `ode_coll2coll` constructor identify the run name and the integer label for the corresponding solution file, stored to disk during the previous run. This time, the desired solution manifold dimensionality is achieved by releasing `'y1e'` and `'y2s'`, while keeping `'T'` fixed and allowing `'y1e'` to vary on the interval $[0, 3]$. The screen output includes the detection of a fold point along the solution manifold, where `'y1e'` is approximately equal to 0.587, as predicted.

We visualize the results of continuation by using the `coco_bd_read` core utility to extract information from the bifurcation data file stored during continuation.

```
>> figure(1); clf; hold on; grid on; box on; axis([0 1 0 3])
>> bd   = coco_bd_read('coll2');
>> labs = coco_bd_labs(bd);
>> for lab=labs
     sol = coll_read_solution('', 'coll2', lab);
     plot(sol.tbp, sol.xbp(:,1), 'LineStyle', '-', 'LineWidth', 2, ...
       'Color', 'black', 'Marker', '.', 'MarkerSize', 12)
   end
>> hold off
```

Here, `bd` is a cell array consisting of a row of string headers and a nonempty collection of subsequent rows of data. The `coco_bd_labs` utility extracts integer labels of solutions stored to disk. The `coll_read_solution` utility reads solution data from each of the corresponding solution files, and assigns a sequence of time instants and corresponding points in state space to the `tbp` and `xbp` fields of the `sol` structure. A single call to the `coco_plot_sol` utility may substitute for the **`for`** loop, as shown here:

```
>> figure(1); clf; hold on; grid on; box on; axis([0 1 0 3])
>> coco_plot_sol('coll2', '')
>> hold off
```

Exercises

1. Try an alternative construction of the initial trajectory segment in the first continuation run in terms of a two-point time history obtained by applying a forward Euler discretization step with step size 0.04 to the point $(x_1, x_2) = (1, 0)$ at $t = 0$.

2. Experiment with the frequency of adaptive remeshing and the initial number of discretization intervals. The latter is set using the `'NTST'` option of the `'coll'` toolbox and defaults to 10.

3. Use the `coco_add_event` utility to add a special point associated with a particular

numerical value of the continuation parameter `'y1e'` and rerun the second continuation run to find the corresponding solution to the two-point boundary-value problem.

4. The method employed in this example for solving the two-point boundary-value problem is commonly referred to as a *shooting method*. Apply the method to the following two-point boundary-value problem

$$y'' + e^{-y} = 0,\ y(0) = 0,\ y(1) = 0$$

from the 2nd edition of Sanchez, D.A., Allen, R.C. Jr., and Kyner, W.T., *Differential Equations*, 1988.

---

3 Heteroclinic connections – `huxley`

As long as $0 < p_1 < 1$ and $p_2 = (1 - 2p_1)/\sqrt{2}$, there exists[4] a connecting orbit of the dynamical system

$$\dot{x}_1 = x_2,\ \dot{x}_2 = p_2 x_2 - x_1(1 - x_1)(x_1 - p_1) \tag{7}$$

between two saddle equilibria at $(x_1, x_2) = (0, 0)$ and $(1, 0)$, given by

$$x_1(t) = \frac{1}{1 + e^{-t/\sqrt{2}}},\ x_2(t) = \frac{1}{\sqrt{2}} \frac{e^{-t/\sqrt{2}}}{(1 + e^{-t/\sqrt{2}})^2}. \tag{8}$$

This orbit coincides with one branch of the unstable manifold of the equilibrium at $(0, 0)$ and one branch of the stable manifold of the equilibrium at $(1, 0)$.

For p_2 away from $(1 - 2p_1)/\sqrt{2}$, the unstable and stable manifolds do not intersect each other, but must each intersect the vertical line $x_1 = p_1$, since i) $\dot{x}_1 > 0$ for $x_2 > 0$ and ii) $\dot{x}_2 > 0$ (< 0) for $x_2 = 0$ and $0 < x_1 < p_1$ ($p_1 < x_1 < 1$). We can use this observation to construct a numerical approximation of the connecting orbit.

To this end, we encode the vector field in the function `huxley` shown below.

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

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

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

end
```

[4]The existence of heteroclinic connections in this *Huxley* model is investigated in Doedel, E.J., and Friedman, M.J., "Numerical computation of heteroclinic orbits," *J. Computational and Applied Mathematics* **26**, pp. 155–170, 1989.

The encoding is vectorized and autonomous. Initial values for the problem parameters and explicit expressions for the relevant stable and unstable eigenvectors are encoded using the following sequence of commands.

```
>> p0   = [0.5; 0];
>> dev0 = [0.03; 0.2];
>> vu   = [sqrt(4*p0(1)+p0(2)^2)-p0(2); 2*p0(1)];
>> vu   = vu/norm(vu, 2);
>> vs   = [-sqrt(4*(1-p0(1))+p0(2)^2)-p0(2); 2*(1-p0(1))];
>> vs   = vs/norm(vs, 2);
```

The `dev0` variable contains initial distances from the equilibria, along the tangent spaces of the unstable and stable manifolds, respectively, to the initial points along the corresponding trajectory segments. The following sequence of assignments stores the corresponding trajectory information in an array of structs, each corresponding to one of the two trajectory segments.

```
>> segs(1).t0 = 0;
>> segs(1).x0 = dev0(1)*vu';
>> segs(1).p0 = p0;
>> segs(2).t0 = 0;
>> segs(2).x0 = [1 0]+dev0(2)*vs';
>> segs(2).p0 = p0;
```

In the following we encode a continuation problem that corresponds to two trajectory segments that satisfy appropriate boundary conditions. As we have reason to perform the construction by repeated continuation under different sets of constraints, we encode the two problem-specific constructors `huxley_isol2het` and `huxley_sol2het` as shown below.

```
function prob = huxley_isol2het(prob, segs, dev0)

prob = ode_isol2coll(prob, 'huxley1', @huxley, ...
  segs(1).t0, segs(1).x0, segs(1).p0);
prob = ode_isol2coll(prob, 'huxley2', @huxley, ...
  segs(2).t0, segs(2).x0, segs(2).p0);

prob = huxley_close_het(prob, dev0);

end


function prob = huxley_sol2het(prob, run, lab)

prob = ode_coll2coll(prob, 'huxley1', run, lab);
prob = ode_coll2coll(prob, 'huxley2', run, lab);

[data chart] = coco_read_solution('bcs', run, lab);
devs = chart.x(data.dev_idx);

prob = huxley_close_het(prob, devs);

end
```

In each of these encodings, the two calls to `ode_isol2coll` or `ode_coll2coll`, respectively, encode a two-segment continuation problem with total dimensional deficit equal to 10. The `huxley_close_het` constructor is shown below.

```
function prob = huxley_close_het(prob, devs)

[data1 uidx1] = coco_get_func_data(prob, 'huxley1.coll', 'data', 'uidx');
[data2 uidx2] = coco_get_func_data(prob, 'huxley2.coll', 'data', 'uidx');

maps1 = data1.coll_seg.maps;
maps2 = data2.coll_seg.maps;

prob = coco_add_glue(prob, 'shared', uidx1(maps1.p_idx), uidx2(maps2.p_idx));

prob = coco_add_func(prob, 'bcs', @huxley_bcs, [], 'zero', 'uidx', ...
  [uidx1(maps1.x0_idx); uidx2(maps2.x1_idx); uidx1(maps1.p_idx)], 'u0', devs);
uidx = coco_get_func_data(prob, 'bcs', 'uidx');
data.dev_idx = [numel(uidx)-1; numel(uidx)];
prob = coco_add_slot(prob, 'bcs', @coco_save_data, data, 'save_full');

prob = coco_add_glue(prob, 'gap', uidx1(maps1.x1_idx(2)), ...
  uidx2(maps2.x0_idx(2)), 'gap', 'inactive');

prob = coco_add_pars(prob, 'pars', ...
  [uidx1(maps1.p_idx); uidx(data.dev_idx); ...
  uidx1(maps1.x1_idx(1)); uidx2(maps2.x0_idx(1))], ...
  {'p1' 'p2' 'dev1' 'dev2' 'y11e' 'y21e'});

end
```

Here, the first call to the `coco_add_glue` utility appends 2 scalar zero problems corresponding to the equality of the problem parameters associated with each of the two trajectory segments. Similarly, the second call to the `coco_add_glue` utility constrains the difference along the x_2 coordinate direction between the final point of the first trajectory segment and the initial point of the second trajectory segment to the value of the inactive continuation parameter `'gap'`. Together, these reduce the dimensional deficit by 3.

The call to the `coco_add_func` utility appends zero functions encoded in the function `huxley_bcs` shown below. These impose the requirement that the initial point on the first trajectory segment and the final point on the second trajectory segment lie on the tangent spaces to the unstable and stable manifolds, respectively, of the corresponding equilibria.

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

x10 = u(1:2);
x20 = u(3:4);
par = u(5:6);
dev = u(7:8);

vu = [sqrt(4*par(1)+par(2)^2)-par(2); 2*par(1)];
vu = vu/norm(vu, 2);
vs = [-sqrt(4*(1-par(1))+par(2)^2)-par(2); 2*(1-par(1))];
vs = vs/norm(vs, 2);
```

```
  y = [x10-dev(1)*vu; x20-([1; 0]+dev(2)*vs)];

  end
```

The call to `coco_add_func` thus adds four equations to the continuation problem, but also appends two more variables to the vector of continuation variables, corresponding to the distances along the two tangent spaces. It follows that the dimensional deficit is reduced by 2. Following this call to `coco_add_func`, the `coco_get_func_data` and `coco_add_slot` utilities are used to extract the construction-independent indices to the additional variables and ensure that these are stored with each solution file for later reference, respectively, as seen in `huxley_sol2het`.

The call to `coco_add_pars` constrains the problem parameters and the deviations, as well as the first components of the final point of the first trajectory segment and the initial point of the second segment to the corresponding continuation parameters `'p1'`, `'p2'`, `'dev1'`, `'dev2'`, `'y11e'`, and `'y21e'`, respectively. It follows that the total dimensional deficit resulting from a call to `huxley_isol2het` or `huxley_sol2het` is -1.

The following sequence of commands constructs an approximation to the heteroclinic connection for the given value of p_1, and then continues this under simultaneous variations in p_1 and p_2.

```
>> prob = huxley_isol2het(coco_prob(), segs, dev0);
>> coco(prob, 'huxley1', [], 1, {'y11e', 'gap'}, [0 0.5]);
>> prob = huxley_sol2het(coco_prob(), 'huxley1', 5);
>> coco(prob, 'huxley2', [], 1, {'y21e', 'gap'}, [0.5 1]);
>> prob = huxley_sol2het(coco_prob(), 'huxley2', 2);
>> coco(prob, 'huxley3', [], 1, {'gap', 'p2'}, [-0.2 0]);
>> prob = huxley_sol2het(coco_prob(), 'huxley3', 4);
>> coco(prob, 'huxley4', [], 1, {'dev1', 'p2'}, [1e-3 dev0(1)]);
>> prob = huxley_sol2het(coco_prob(), 'huxley4', 3);
>> coco(prob, 'huxley5', [], 1, {'dev2', 'p2'}, [1e-3 dev0(2)]);
>> prob = huxley_sol2het(coco_prob(), 'huxley5', 3);
>> coco(prob, 'huxley6', [], 1, {'p1', 'p2'}, [0.25 0.75]);
```

Specifically, in the first continuation run, we release `'y11e'` and `'gap'` and allow these to vary within the given computational domain. The value of 0.5 for `'y11e'` corresponds to a solution with the final point of the first trajectory segment on the vertical line $x_1 = p_1$. In the second continuation run, we start from this solution and allow `'y21e'` and `'gap'` to vary within the given computational domain. In this case, the value of 0.5 for `'y21e'` corresponds to a solution with the initial point of the second trajectory segment on the vertical line $x_1 = p_1$. We reduce the gap between the two points on $x_1 = p_1$ to zero in the third continuation run. In the following two continuation runs, we reduce the distances, along the associated tangent spaces, from the initial point of the first trajectory segment and the final point of the second segment to the corresponding equilibria.

Exercises

1. Use the two-segment approximation to the heteroclinic connection to construct an initial solution guess for a single-segment continuation problem with appropriate boundary conditions. Comment on the utility of the approach taken in the example and an alternative reliance on a single-segment continuation problem for all parts of the analysis.

2. Use the `coco_bd_read` and `coco_bd_col` utilities to extract the values of `'p1'` and `'p2'` from the bifurcation data cell array stored during the final continuation run. Graph the corresponding relationship between the problem parameters and compare this to the theoretical prediction.

3. Use the `coll_read_solution` utility to extract the trajectory segments for each labeled solution and generate an animation showing the sequence of consecutive solutions to the extended continuation problem that lead to the construction of the two-segment approximation to the heteroclinic connection. You may wish to modify the `'NPR'` setting of the `'cont'` toolbox to increase the frequency of storing solutions to disk. You may also wish to modify the computational domains used in each continuation run to restrict continuation to the desired direction along the corresponding solution manifold.

4. Implement the continuation of heteroclinic connections in the planar vector field

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

analyzed in Doedel, E.J. and Friedman, M.J., "Numerical computation of heteroclinic orbits," *J. Computational and Applied Mathematics* **26**, pp. 155–170, 1989. Use the methodology described in Sect. 7.3.3 of *Recipes for Continuation* by combining instances of the `'ep'` and `'coll'` toolboxes (cf. the `doedel` demo).

---

4 Harmonic excitation – `linode`

Consider the linear oscillator with harmonic excitation governed by the explicitly time-dependent dynamical system

$$\dot{x}_1 = x_2,\ \dot{x}_2 = -px_1 - x_2 + \cos t \tag{9}$$

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 arbitrary initial conditions, the steady-state behavior is then given by the 2π-periodic orbit

$$x_1(t) = x_1^*(t) := \frac{\sin t + (p-1)\cos t}{p^2 - 2p + 2},\ x_2(t) = x_2^*(t) := \frac{\cos t - (p-1)\sin t}{p^2 - 2p + 2} \tag{10}$$

with $\mathcal{L}_2$ norm

$$\|x^*\|_2 := \sqrt{\int_0^{2\pi} \left(x_1^{*2}(t) + x_2^{*2}(t)\right) \mathrm{d}t} = \frac{\sqrt{2\pi}}{\sqrt{p^2 - 2p + 2}}. \tag{11}$$

If we let

$$x_1(t) = x_1^*(t) + y_1(t),\ x_2(t) = x_2^*(t) + y_2(t) \tag{12}$$

then the dynamical system

$$\dot{y}_1 = y_2,\ \dot{y}_2 = -py_1 - y_2 \tag{13}$$

is the variational equation about the periodic steady-state trajectory. For the initial conditions $y_1(0) = 1$ and $y_2(0) = 0$, we obtain the solution

$$y_1(t) = y_{11}(t) := e^{-t/2}\left(\cosh\left(\frac{\sqrt{1-4p}}{2}t\right) + \frac{1}{\sqrt{1-4p}}\sinh\left(\frac{\sqrt{1-4p}}{2}t\right)\right) \tag{14}$$

and

$$y_2(t) = y_{21}(t) := -\frac{2pe^{-t/2}}{\sqrt{1-4p}}\sinh\left(\frac{\sqrt{1-4p}}{2}t\right). \tag{15}$$

Similarly, when $y_1(0) = 0$ and $y_2(0) = 1$, we obtain the solution

$$y_1(t) = y_{12}(t) := \frac{2e^{-t/2}}{\sqrt{1-4p}}\sinh\left(\frac{\sqrt{1-4p}}{2}t\right) \tag{16}$$

and

$$y_2(t) = y_{22}(t) := e^{-t/2}\left(\cosh\left(\frac{\sqrt{1-4p}}{2}t\right) - \frac{1}{\sqrt{1-4p}}\sinh\left(\frac{\sqrt{1-4p}}{2}t\right)\right). \tag{17}$$

The fundamental solution matrix

$$\Phi(t) := \begin{pmatrix} y_{11}(t) & y_{12}(t) \\ y_{21}(t) & y_{22}(t) \end{pmatrix} \tag{18}$$

satisfies the variational equation, and every solution to this equation may be expressed as $\Phi(t)C$ where the column matrix C contains the initial conditions for y_1 and y_2.

The value of the fundamental solution matrix at $t = 2\pi$ is the *monodromy matrix*

$$M := \Phi(2\pi) = \frac{e^{-\pi}}{\tilde{p}}\begin{pmatrix} \tilde{p}\cosh(\tilde{p}\pi) + \sinh(\tilde{p}\pi) & 2\sinh(\tilde{p}\pi) \\ -2p\sinh(\tilde{p}\pi) & \tilde{p}\cosh(\tilde{p}\pi) - \sinh(\tilde{p}\pi) \end{pmatrix}, \tag{19}$$

where $\tilde{p} = \sqrt{1-4p}$. This is the matrix of perturbations to the final point on the solution trajectory. Its columns correspond to unit perturbations to the initial point along each of the coordinate directions. The eigenvalues of the monodromy matrix are the *Floquet multipliers* of the periodic orbit. The periodic orbit is Poincaré stable provided that these lie within the unit circle in the complex plane. In the present case, the Floquet multipliers equal

$$e^{(-1\pm\tilde{p})\pi} \tag{20}$$

and remain within the unit circle for all values of p, as expected.

We encode vectorized implementations of the vector field and its Jacobians with respect to the state variables, parameters, and time in the functions `linode_het`, `linode_het_DFDX`, `linode_het_DFDP`, and `linode_het_DFDT` shown below.

```
function y = linode_het(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_het_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_het_DFDP(t, x, p)

x1 = x(1,:);

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

end


function J = linode_het_DFDT(t, x, p)

x1 = x(1,:);

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

end
```

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

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

The following sequence of commands encodes a trajectory segment continuation problem

using the ode_isol2coll constructor.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'autonomous', false);
>> prob = coco_set(prob, 'coll', 'NTST', 15);
>> prob = coco_set(prob, 'cont', 'NAdapt', 1);
>> coll_args = {@linode_het, @linode_het_DFDX, @linode_het_DFDP, ...
     @linode_het_DFDT, t0, x0, 'p', 1};
>> prob = ode_isol2coll(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, and is allowed to change after each successful step of continuation as a consequence of an adaptive remeshing of the discretization. The dimensional deficit of the continuation problem encoded thus far in prob equals the number of state-space dimensions plus 2, i.e., 4.

We compute a family of periodic orbits under variations in p by invoking the coco entry-point function as shown in the sequence of commands below.

```
>> [data uidx] = coco_get_func_data(prob, 'coll', 'data', 'uidx');
>> maps = data.coll_seg.maps;
>> prob = coco_add_func(prob, 'po', @linode_het_bc, data, 'zero', ...
     'uidx', uidx([maps.x0_idx; maps.x1_idx; maps.T0_idx; maps.T_idx]));
>> coco(prob, 'het_run1', [], 1, {'p' 'coll.err_TF'}, [0.2 2]);
```

Here, the coco_get_func_data utility extracts the function data structure and the function dependency index set for the instance of the 'coll' toolbox constructed previously. As in the previous section, the data.coll_seg.maps field contains construction-independent integer indices for distinct elements in the vector of continuation variables that can be used to refer to the variables representing the initial and final points on the trajectory segment, as well as to the initial time and the interval length. The subindexing

```
uidx([maps.x0_idx; maps.x1_idx; maps.T0_idx; maps.T_idx])
```

constructs a vector of integers referencing the corresponding elements of the vector of continuation variables. These constitute the components of the u input argument to the function linode_het_bc whose encoding is shown below.

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

x0 = u(1:2);
x1 = u(3:4);
T0 = u(5);
T  = u(6);

y = [x1(1:2)-x0(1:2); T0; T-2*pi];

end
```

Since this is added to the continuation problem as a zero function, it follows that a solution corresponds to a closed curve in state space with interval length 2π. The dimensional deficit

of this 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 = coco_set(prob, 'cont', 'NAdapt', 1);
>> prob = ode_coll2coll(prob, '', 'het_run1', 3);
>> [data uidx] = coco_get_func_data(prob, 'coll', 'data', 'uidx');
>> maps = data.coll_seg.maps;
>> prob = coco_add_func(prob, 'po', @linode_het_bc, data, 'zero', ...
     'uidx', uidx([maps.x0_idx; maps.x1_idx; maps.T0_idx; maps.T_idx]));
>> coco(prob, 'het_run2', [], 1, {'p' 'coll.err_TF'}, [0.2 2]);
```

These commands differ from the previous construction only in the use of the `ode_coll2coll` constructor. As an alternative, we may restart continuation from one of the periodic orbits obtained in a previous run *and* simultaneously continue one or several solutions to the corresponding variational equation. The latter describes the linear sensitivity of the periodic orbit to particular perturbations, and provides a method for analyzing the orbit stability. Notably, the variational equation is automatically encoded by the `'coll'` toolbox, given the original vector field and, if available, its Jacobian with respect to the state.

As an example, the following sequence of commands uses the `ode_coll2coll` constructor to append two copies of the variational equation with initial solution guesses corresponding to unit perturbations of the initial point on the solution trajectory along the x_1 and x_2 coordinate direction, respectively.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'autonomous', false);
>> prob = coco_set(prob, 'coll', 'NTST', 25);
>> prob = ode_coll2coll(prob, '', 'het_run1', 3, '-var', eye(2));
>> [data uidx] = coco_get_func_data(prob, 'coll', 'data', 'uidx');
>> maps = data.coll_seg.maps;
>> prob = coco_add_func(prob, 'po', @linode_het_bc, data, 'zero', ...
     'uidx', uidx([maps.x0_idx; maps.x1_idx; maps.T0_idx; maps.T_idx]));
```

In this case, no adaptive remeshing is deployed during continuation, but the number of discretization intervals is set to 25 in order to maintain an acceptable discretization error estimate. As long as the perturbations to the initial point are held fixed during continuation, the combined solution to the two variational equations is the fundamental solution for the linearized problem about the periodic orbit. This is accomplished by the following commands.

```
>> [data uidx] = coco_get_func_data(prob, 'coll.var', 'data', 'uidx');
>> prob = coco_add_pars(prob, 'pars', uidx(data.coll_var.v0_idx(:)), ...
     {'s1' 's2' 's3' 's4'});
```

Here, the function data structure and function dependency index set for the variational

equation zero functions are extracted using the `coco_get_func_data` utility. The subindexing `uidx(data.coll_var.v0_idx(:))` returns a sequence of integers corresponding to the perturbations to the initial solution point.

We invoke the `coco` entry-point function to perform simultaneous continuation of the periodic orbit and the corresponding fundamental solution.

```
>> coco(prob, 'het_run_var', [], 1, {'p' 'coll.err_TF'}, [0.05 3]);
>> data  = coco_read_solution('coll', 'het_run_var', 2, 'data');
>> chart = coco_read_solution('coll.var', 'het_run_var', 2, 'chart');
>> M = chart.x(data.coll_var.v1_idx);
>> p = chart.x(data.coll_seg.maps.p_idx);
>> sort([eig(M) [exp(-pi+sqrt(1-4*p)*pi); exp(-pi-sqrt(1-4*p)*pi)]])
```

Here, the `coco_read_solution` utility extracts the stored solution data and part of the solution chart associated with the variational equations from the second labeled solution found during the previous run. Next, the monodromy matrix is assigned to the variable `M` and the corresponding value of the problem parameter p is assigned to the variable `p`. Finally, the eigenvalues of the monodromy matrix obtained using continuation are compared to their theoretical values.

Exercises

1. Use the `ode_isol2coll` constructor to continue simultaneously a periodic orbit and the solutions to the variational equation corresponding to three different initial conditions. Compare the values at 2π with the theoretical values obtained from the product
$$M \cdot \begin{pmatrix} x_1(0) \\ x_2(0) \end{pmatrix}$$

2. Use the `coll_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.

3. The `'coll'` toolbox saves the $\mathcal{L}_2$ norm associated with each solution trajectory in the `'||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.

4. Repeat the theoretical and computational analysis in this section for the equivalent autonomous vector field
$$\dot{x}_1 = x_2,\ \dot{x}_2 = -px_1 - x_2 + \cos x_3,\ \dot{x}_3 = 1$$
on $\mathbb{R}^2 \times \mathbb{S}^1$, where x_3 is an angle parameter on $\mathbb{S}^1$.

---

5 Bratu's boundary-value problem – `bratu`

The `ode_isol2coll` and `ode_coll2coll` constructors encode a collocation continuation problem for a single trajectory segment with no additional constraints on the trajectory segment. In a boundary-value problem, one or several trajectory segments may be further constrained by the imposition of conditions on the segment end points. In previous examples, such conditions were introduced separately using the `coco_add_func` constructor. In this and the next sections, we demonstrate the use of the `ode_isol2bvp` and `ode_bvp2bvp` constructors to accomplish this task.

Consider, for example, the boundary-value problem

$$\dot{x}_1 = x_2,\ \dot{x}_2 = -pe^{x_1},\ x_1(0) = 0,\ x_1(1) = 0 \tag{21}$$

in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and the scalar problem parameter $p \in \mathbb{R}$. A solution to this boundary-value problem is of the form

$$x_1(t) = \ln\left(\frac{1+\cosh C}{1+\cosh\left(C(1-2t)\right)}\right),\ x_2(t) = 2C\tanh\left(\frac{C}{2}(1-2t)\right) \tag{22}$$

provided that

$$p = \frac{4C^2}{1+\cosh C}. \tag{23}$$

For $C \geq 0$, the right-hand side of (23) has a unique global maximum at $C = C^* \approx 2.399$, corresponding to $p = p^* \approx 3.514$, and decays to 0 as $C \to \infty$. It follows that there exist two solutions of the given form to the boundary-value problem for $0 < p < p^*$ and none for $p > p^*$. When $p = 0$, a corresponding solution is given by $(x_1(t), x_2(t)) = (0, 0)$ for all t.

We construct a family of solutions to the boundary-value problem for admissible values of p by starting continuation with the explicit solution for $p = 0$. To this end, we encode the vector field in the anonymous function `brat`.

```
>> brat = @(x,p) [x(2,:); -p(1,:).*exp(x(1,:))];
```

The encoding is vectorized and autonomous. We encode the boundary conditions and their derivatives with respect to the interval length, the coordinates of the two end points, and the problem parameters in the anonymous functions `brat_bc` and `brat_bc_DFDX`, respectively.

```
>> brat_bc      = @(~,T,x0,x1,p) [T-1; x0(1); x1(1)];
>> brat_bc_DFDX = @(~,T,x0,x1,p) [1,0,0,0,0,0; 0,1,0,0,0,0; 0,0,0,1,0,0];
```

In particular, we require that the interval length T equal 1 and that the first component of each of the end points equal 0. The Jacobian encoded in `brat_bc_DFDX` is a 3×6 two-dimensional array, since the number of scalar boundary conditions is 3 and the number of arguments equals $6 (= 1 + 2 + 2 + 1)$. The `~` in the first input argument is a placeholder for data specific to the encoding of the boundary conditions.

We compute a family of solutions to the boundary-value problem under variations in p by constructing a constrained single-segment continuation problem using the `ode_isol2bvp`

constructor, and invoking the coco entry-point function as shown in the sequence of commands below.

```
>> prob      = coco_prob();
>> prob      = coco_set(prob, 'cont', 'PtMX', 50);
>> coll_args = {brat, [0;1], zeros(2), 0};
>> bvp_args  = [coll_args, 'p', {brat_bc, brat_bc_DFDX} ];
>> bd = coco(prob, 'brat1', @ode_isol2bvp, bvp_args{:}, 1, 'p', [0 4]);
```

Here, the coll_args variable contains the input argument to a single embedded call to the ode_isol2coll constructor. Notably, this omits the parameter name 'p', which is inserted into the bvp_args variable after coll_args and before a cell array containing the function names of the encodings of the boundary conditions and their Jacobian. Since the number of boundary conditions equals the dimensional deficit of the trajectory segment continuation problem, the total dimensional deficit of the continuation problem constructed by ode_isol2bvp is 0. The screen output includes the detection of a fold point along the solution manifold, where 'p' is approximately equal to 3.514, as predicted.

We visualize the result of continuation by extracting information from the bifurcation data file stored during continuation using the coco_bd_read core utility.

```
>> labs = coco_bd_labs(bd);
>> figure(1); clf; hold on; grid on; box on; axis([0 1 -0.1 1.5])
>> for lab=labs
     sol = bvp_read_solution('', 'brat1', lab);
     plot(sol{1}.tbp, sol{1}.xbp(:,1), 'LineStyle', '-', 'LineWidth', 2, ...
        'Color', 'black', 'Marker', '.', 'MarkerSize', 12)
   end
>> hold off
```

Here, bd is a cell array consisting of a row of string headers and a nonempty collection of subsequent rows of data. The coco_bd_labs utility extracts integer labels of solutions stored to disk. The bvp_read_solution utility reads solution data from each of the corresponding solution files, and assigns a sequence of time instants and corresponding points in state space to the tbp and xbp fields of the first element of the sol structure array.

As shown in the following sequence of commands, we can restart continuation from a solution stored to disk during the previous continuation run.

```
>> prob = coco_prob();
>> prob = coco_set(prob, 'cont', 'NAdapt', 5);
>> coco(prob, 'brat2', @ode_bvp2bvp, 'brat1', 6, 1, 'p', [1 5]);
```

Here, the two arguments following the @ode_bvp2bvp function handle identify the run name and integer label for the corresponding solution file. The call to the coco_set utility ensures an adaptive remeshing of the trajectory discretization after every five successful steps of continuation.

Exercises

1. Use the `ode_isol2bvp` constructor to build the continuation problem corresponding to the linear oscillator in the previous section. Try both the autonomous and nonautonomous encodings of the vector field and include explicit Jacobians.

2. Perform continuation on the domain $C \in [0, 5]$ of solutions to the boundary-value problem
$$\dot{x}_1 = x_2,\ \dot{x}_2 = -\frac{4C^2}{1+\cosh C}e^{x_1},\ x_1(0) = 0,\ x_1(1) = 0$$
using as initial solution guess a two-point discretization of $(x_1(t), x_2(t)) = (0, 0)$ for all t when $C = 0$. Make appropriate changes to the `'NTST'` and `'NAdapt'` settings of the `'coll'` and `'cont'` toolboxes to avoid termination due to excessive discretization errors.

3. Restart continuation using the `ode_BP2bvp` constructor from the branch point detected in the previous exercise. What happens when $C \to 0$? Can you find an explicit expression for $x_1(t)$ and $x_2(t)$ along this second branch?

4. Use continuation to investigate solutions to the boundary-value problem for $p < 0$.

---

6 A moving Poincaré section – `lienard`

Consider the autonomous dynamical system
$$\dot{x}_1 = x_2,\ \dot{x}_2 = px_2 - x_2^3 - x_1 \tag{24}$$
in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and the scalar problem parameter $p \in \mathbb{R}$. This is encoded in the function `lienard` shown below.

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

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

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

end
```

As long as $p \geq 0$, there exists a periodic orbit enclosing the origin. In contrast to the linear oscillator considered in a previous section, the orbit period is unknown. Moreover, periodic boundary conditions do not uniquely identify the location of the trajectory segment end points along the corresponding closed curve in the plane.

A unique parameterization of the periodic orbit is obtained by imposing a Poincaré section condition. Specifically, we require that the initial point on the trajectory segment lie on a hyperplane through some reference point and perpendicular to the vector field evaluated at the reference point. We encode the corresponding boundary conditions and their Jacobian in the functions `per_bc` and `per_bc_DFDX` shown below.

```
function fbc = per_bc(data, T, x0, x1, p)
  fbc = [x0-x1; data.f0*(x0-data.x0)];
end


function Jbc = per_bc_DFDX(data, T, x0, x1, p)
  Jbc = data.J;
end
```

Notably, both encodings rely on the content of the `data` input argument, which must be populated and, as necessary, updated during continuation. To this end, consider the function `per_bc_update` shown below.

```
function data = per_bc_update(data, T, x0, x1, p)

n = numel(x0);
q = numel(p);

data.x0 = x0;
data.f0 = data.fhan(x0,p)';
data.J  = [sparse(n,1), speye(n,n), -speye(n,n), sparse(n,q);
           sparse(1,1), data.f0,    sparse(1,n), sparse(1,q)];

end
```

This assigns content to the `x0`, `f0`, and `J` fields of the `data` output argument corresponding to a parameterization of the hyperplane in terms of the point `x0` in the second input argument. The following sequence of commands then initializes the Poincaré section.

```
>> p0 = 1;
>> x0 = [0.4; -1.2];
>> data = struct();
>> data.fhan = @lienard;
>> data = per_bc_update(data, [], x0, [], p0);
```

We proceed to construct the corresponding boundary-value problem, as shown in the following sequence of commands.

```
>> f = @(t,x) lienard(x, p0);
>> [t0 x0]   = ode45(f, [0 6.7], x0);
>> coll_args = { @lienard, t0, x0, p0 };
>> bvp_args  = { @per_bc, @per_bc_DFDX, data, @per_bc_update };
>> prob = coco_prob();
>> prob = coco_set(prob, 'coll', 'NTST', 30);
>> prob = ode_isol2bvp(prob, '', coll_args{:}, 'p', bvp_args{:});
```

Here, the inclusion of `data` and the `@per_bc_update` function handles in the `bvp_args` input

argument ensures that the content of `data` is available to `per_bc` and `per_bc_DFDX`, and updated before each new continuation step. Specifically, the reference point of the Poincaré section, stored in `data.x0` equals the initial point along the trajectory segment associated with the current solution from which continuation proceeds.

Continuation along a family of periodic orbits with adaptive remeshing every ten continuation steps is achieved using the following commands.

```
>> prob = coco_set(prob, 'cont', 'NAdapt', 10);
>> bd = coco(prob, 'moving', [], 1, 'p', [-1 1]);
```

We visualize individual labeled solutions, as well as the location of the reference points of the sequence of Poincaré sections using the following commands.

```
>> labs = coco_bd_labs(bd);
>> x0 = [];
>> figure(1); clf; hold on; grid on; box on; axis([-1.75 1.75 -1.4 1.23])
>> for lab=labs
     sol = bvp_read_solution('', 'moving', lab);
     plot(sol{1}.xbp(:,1), sol{1}.xbp(:,2), 'LineStyle', '-', 'LineWidth', 2, ...
       'Color', [0.7 0.7 0.7], 'Marker', '.', 'MarkerSize', 12)
     x0 = [x0 ; sol{1}.xbp(1,:)];
   end
>> plot(x0(:,1), x0(:,2), 'LineStyle', 'none', 'LineWidth', 2, ...
     'Color', 'black', 'Marker', '.', 'MarkerSize', 15)
>> hold off
```

Here, the `bvp_read_solution` utility extracts the individual trajectory segments and stores the corresponding discretization in `sol`.

Exercises

1. Repeat the construction of the boundary-value problem, but omit the inclusion of the `per_bc_update` function. Visualize the result of continuation and describe your observations.

2. Suppose that continuation along a one-dimensional family of periodic orbits includes the occurrence of a tangency with a fixed Poincaré section. Explain why this manifests itself as a geometric fold along the solution manifold.

3. Apply the `ode_isol2bvp` and `ode_bvp2bvp` constructors to the continuation of periodic orbits of the dynamical system

$$\dot{x}_1 = x_2,\ \dot{x}_2 = \left(\frac{1}{2} - x_2^2\right)\varepsilon x_2 - x_1$$

   under variations in ε on the computational domain $[-10, 10]$. Graph the variations in the corresponding Floquet multipliers against ε.

---

7 Invariant curves and tori – `torus`

Consider the non-autonomous dynamical system $\dot{x} = F(t, x, p)$, where

$$F(t,x,p) = \begin{pmatrix} -\Omega x_2 + x_1\Big(1 + \sqrt{x_1^2 + x_2^2}(\cos\omega t - 1)\Big) \\ \\ \Omega x_1 + x_2\Big(1 + \sqrt{x_1^2 + x_2^2}(\cos\omega t - 1)\Big) \end{pmatrix} \tag{25}$$

in terms of the vector of state variables $x = (x_1, x_2) \in \mathbb{R}^2$ and the vector of problem parameters $(\omega, \Omega) \in \mathbb{R}^2$. In polar coordinates (ρ, ψ), where $x_1 = \rho\cos\psi$ and $x_2 = \rho\sin\psi$, it follows that

$$\dot{\rho} = \rho\big(1 + \rho(\cos\omega t - 1)\big),\ \dot{\psi} = \Omega \tag{26}$$

and, consequently, that

$$\rho(t) = \frac{e^t\rho_0(1+\omega^2)}{1 + \omega^2 - \omega^2\rho_0 + e^t\rho_0\big(1 + \omega^2 - \cos\omega t - \omega\sin\omega t\big)},\ \psi(t) = \Omega t + \psi_0 \tag{27}$$

in terms of the initial conditions (ρ_0, ψ_0). In particular, for $t \gg 1$,

$$\rho(t) \approx \rho^*(t) := \frac{1+\omega^2}{1 + \omega^2 - \cos\omega t - \omega\sin\omega t} \tag{28}$$

corresponding to motion on an invariant two-dimensional torus $\mathbb{T}$ described by the torus function $u : \mathbb{S}\times\mathbb{S} \to \mathbb{T}$, where

$$u : (\theta_1, \theta_2) \mapsto \big(\rho^*(\theta_2/\omega)\cos\theta_1, \rho^*(\theta_2/\omega)\sin\theta_1\big), \tag{29}$$

and by the two frequencies Ω and ω, such that the dynamics on the torus correspond to

$$\frac{d\theta_1}{dt} = \Omega, \text{ and } \frac{d\theta_2}{dt} = \omega. \tag{30}$$

In particular,

$$\Omega\frac{\partial u}{\partial\theta_1}(\theta_1,\theta_2) + \omega\frac{\partial u}{\partial\theta_2}(\theta_1,\theta_2) = F(\theta_2/\omega, u(\theta_1,\theta_2), p). \tag{31}$$

For a given rotation number $\varrho = \Omega/\omega$, the dynamics on the torus is a parallel flow, consisting of either i) torus-covering quasiperiodic trajectories, in the case that ϱ is irrational, or ii) a continuous family of periodic orbits, in the case that ϱ is rational.

The definition $v(\varphi, \tau) := u(\varphi + \Omega\tau, \omega\tau)$ implies that

$$v(\varphi, 0) = u(\varphi, 0),\ v(\varphi, 2\pi/\omega) = u(\varphi + 2\pi\varrho, 0), \text{ and } \frac{\partial v}{\partial\tau} = F(\tau, v(\varphi,\tau), p). \tag{32}$$

Here, $u(\varphi, 0)$ is the circle in $\mathbb{R}^2$ centered at the origin and with radius $(1+\omega^2)/\omega^2$ parameterized by $\varphi \in [0, 2\pi]$. It follows that this circle is invariant under the mapping from $\tau = 0$ to $\tau = 2\pi/\omega$. On this invariant circle, the mapping is a rigid rotation by $2\pi\varrho$.

We may approximate a component of $\upsilon(\varphi,0)$ by a truncated Fourier expansion

$$\chi(\varphi) = a_0 + \sum_{k=1}^{N} \Big(a_k \cos k\varphi + b_k \sin k\varphi \Big), \tag{33}$$

where

$$\begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ \vdots \\ a_N \\ b_N \end{pmatrix} = \mathcal{F} \cdot \begin{pmatrix} \chi(0) \\ \chi\left(\frac{2\pi}{2N+1}\right) \\ \vdots \\ \chi\left(\frac{4\pi N}{2N+1}\right) \end{pmatrix} \tag{34}$$

in terms of the discrete Fourier transform matrix

$$\mathcal{F} = \frac{1}{2N+1} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 2 & 2\cos\left(\frac{2\pi}{2N+1}\right) & \cdots & 2\cos\left(\frac{4\pi N}{2N+1}\right) \\ 0 & 2\sin\left(\frac{2\pi}{2N+1}\right) & \cdots & 2\sin\left(\frac{4\pi N}{2N+1}\right) \\ \vdots & \vdots & \ddots & \vdots \\ 2 & 2\cos\left(\frac{2\pi N}{2N+1}\right) & \cdots & 2\cos\left(\frac{4\pi N^2}{2N+1}\right) \\ 0 & 2\sin\left(\frac{2\pi N}{2N+1}\right) & \cdots & 2\sin\left(\frac{4\pi N^2}{2N+1}\right) \end{pmatrix}. \tag{35}$$

On the other hand,

$$\chi(\varphi + 2\pi\varrho) = a'_0 + \sum_{k=1}^{N} \Big(a'_k \cos k\varphi + b'_k \sin k\varphi \Big) \tag{36}$$

implies that

$$\begin{pmatrix} a'_0 \\ a'_1 \\ b'_1 \\ \vdots \\ a'_N \\ b'_N \end{pmatrix} = \mathcal{R} \cdot \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ \vdots \\ a_N \\ b_N \end{pmatrix} \tag{37}$$

in terms of the rotation matrix

$$\mathcal{R} = \begin{pmatrix} 1 & & & & & \\ & \cos 2\pi\varrho & \sin 2\pi\varrho & & & \\ & -\sin 2\pi\varrho & \cos 2\pi\varrho & & & \\ & & & \ddots & & \\ & & & & \cos 2\pi N\varrho & \sin 2\pi N\varrho \\ & & & & -\sin 2\pi N\varrho & \cos 2\pi N\varrho \end{pmatrix}. \tag{38}$$

On the mesh $\varphi_j := 2\pi(j-1)/(2N+1)$, it follows that

$$\left(\mathcal{F} \otimes I_2\right) \cdot \begin{pmatrix} \upsilon\left(\varphi_1, 2\pi/\omega\right) \\ \vdots \\ \upsilon\left(\varphi_{2N+1}, 2\pi/\omega\right) \end{pmatrix} = \left(\left(\mathcal{R} \cdot \mathcal{F}\right) \otimes I_2 \right) \cdot \begin{pmatrix} \upsilon\left(\varphi_1, 0\right) \\ \vdots \\ \upsilon\left(\varphi_{2N+1}, 0\right) \end{pmatrix}. \tag{39}$$

and

$$\frac{d\upsilon}{d\tau}\left(\varphi_j, \tau\right) = F\left(\omega\tau, \upsilon\left(\varphi_j, \tau\right), p\right) \tag{40}$$

for $j = 1, \ldots, 2N+1$. We may eliminate the degeneracy associated with arbitrary shifts in φ by demanding that $\upsilon_2(0,0) = 0$.

We proceed to perform continuation of an approximate Fourier representation of the function $\upsilon(\varphi, 0)$. The vector field and its Jacobians with respect to the state variables, the problem parameters, and time are encoded in the functions `torus`, `torus_DFDX`, `torus_DFDP`, and `torus_DFDT` shown below.

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

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

r = sqrt(x1.^2+x2.^2);
y(1,:) = -Om.*x2+x1.*(1+r.*(cos(om.*t)-1));
y(2,:) = Om.*x1+x2.*(1+r.*(cos(om.*t)-1));

end


function J = torus_DFDX(t, x, p)

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

r = sqrt(x1.^2+x2.^2);
J = zeros(2,2,numel(x1));
J(1,1,:) = (r+(r.^2+x1.^2).*(cos(om.*t)-1))./r;
J(1,2,:) = (-Om.*r+x1.*x2.*(cos(om.*t)-1))./r;
J(2,1,:) = (Om.*r+x1.*x2.*(cos(om.*t)-1))./r;
J(2,2,:) = (r+(r.^2+x2.^2).*(cos(om.*t)-1))./r;

end


function J = torus_DFDP(t, x, p)

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

```
r = sqrt(x1.^2+x2.^2);
J = zeros(2,2,numel(x1));
J(1,1,:) = -r.*t.*x1.*sin(om.*t);
J(1,2,:) = -x2;
J(2,1,:) = -r.*t.*x2.*sin(om.*t);
J(2,2,:) = x1;

end


function J = torus_DFDT(t, x, p)

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

r = sqrt(x1.^2+x2.^2);
J = zeros(2,numel(x1));
J(1,:) = -x1.*r.*om.*sin(om.*t);
J(2,:) = -x2.*r.*om.*sin(om.*t);

end
```

We initialize a cell array of input arguments for `ode_isol2coll` in the following sequence of commands.

```
>> om   = 1.5;
>> Om   = 1;
>> N    = 15;
>> vphi = 2*pi*linspace(0,1,2*N+2);
>> tau  = 2*pi/om*linspace(0,1,10*(2*N+1))';
>> rho  = (1+om^2)./(1+om^2-cos(om*tau)-om*sin(om*tau));
>> coll = cell(1,2*N+1);
>> for i=1:2*N+1
     up = repmat(rho, [1 2]).*[cos(Om*tau+vphi(i)) sin(Om*tau+vphi(i))];
     coll{i} = {@torus @torus_DFDX @torus_DFDP @torus_DFDT tau up [om Om]};
   end
```

Here, the variable `rho` contains an evenly spaced sample of values of $\rho^*(\tau)$ for τ on the interval $[0, 2\pi/\omega]$. For each value of φ, the corresponding values for υ are stored in `up`.

The discrete Fourier transform and rotation matrices $\mathcal{F}$ and $\mathcal{R}$ are constructed in the following sequence of commands.

```
>> Th = 2*pi*(0:2*N)/(2*N+1);
>> Th = kron(1:N, Th');
>> F  = [ones(2*N+1,1) 2*reshape([cos(Th);sin(Th)], [2*N+1 2*N])]'/(2*N+1);
>> varrho = 1/1.51111;
>> Th  = (1:N)*2*pi*varrho;
>> SIN = [ zeros(size(Th)) ; sin(Th) ];
>> R   = diag([1 kron(cos(Th), [1 1])]);
>> R   = R  + diag(SIN(:), +1)- diag(SIN(:), -1);
```

We store $\mathcal{F}\otimes I_2$ and $(\mathcal{R}\cdot\mathcal{F})\otimes I_2$ in the variable `data` and proceed to construct the appropriate constrained multisegment boundary-value problem using the `ode_isol2bvp` constructor, as

shown below.

```
>> data    = struct();
>> data.F  = kron(F, eye(2));
>> data.RF = kron(R*F, eye(2));
>> prob = coco_prob();
>> prob = coco_set(prob, 'ode', 'autonomous', false);
>> prob = coco_set(prob, 'coll', 'NTST', 20);
>> prob = ode_isol2bvp(prob, '', coll, {'om' 'Om'}, @torus_bc, data, 'F+dF');
```

Here, the `'F+dF'` option is invoked to indicate that the function encoding the boundary conditions also returns their Jacobian, as shown in the encoding of `torus_bc` below.

```
function [fbc, Jbc] = torus_bc(data, T0, T, x0, x1, p)

fbc = [T0; T-2*pi/p(1); data.F*x1-data.RF*x0; x0(2)];

nt = numel(T);
nx = numel(x0);
np = numel(p);

J1 = zeros(1,2*nt+2*nx+np);
J1(1,2*nt+2) = 1;

Jbc = [
  eye(nt), zeros(nt,nt+2*nx+np);
  zeros(nt), eye(nt), zeros(nt,2*nx), 2*pi/p(1)^2*ones(nt,1), zeros(nt,np-1);
  zeros(nx,2*nt), -data.RF, data.F, zeros(nx,np);
  J1];
end
```

Notably, the rotation number ϱ is not included among the continuation variables, so continuation results in a family of discrete Fourier representations for fixed ratio Ω/ω, as shown by executing the following commands.

```
>> prob = coco_set(prob, 'cont', 'NAdapt', 1, 'h_max', 10);
>> coco(prob, 'torus', [], 1, {'om' 'Om'}, [0.5 1.5]);
```

Exercises

1. Use the explicit time history in (27) to encode an algebraic continuation problem that is equivalent to the constrained multisegment boundary-value problem in this section, but that eliminates the need to invoke the `'coll'` toolbox for the approximate discretization of individual solution trajectories. Compare the results of continuation using the two alternative constructions.

2. Use $\varrho = 1.5$ in the construction of the constrained multisegment boundary-value problem and explore the dependence on N. Explain your observations by comparing and contrasting with the behavior of the algebraic implementation in the previous exercise.

3. Construct a continuation problem in which the rotation number ϱ is included among the continuation variables.

4. Repeat the construction of a constrained multisegment boundary-value problem for the continuation of invariant tori corresponding to the vector field

$$F(t,x,p) = \begin{pmatrix} x_2 \\ cx_2(1-x_1^2) - x_1 + a\cos\omega t \end{pmatrix}.$$

---

8 Optimization – `linode_optim`

Consider the problem of finding stationary points of the functional $(x(t),k,\theta) \mapsto x_2(0)$ along a manifold of periodic solutions of the dynamical system

$$\dot{x}_1 = x_2,\ \dot{x}_2 = -x_2 - kx_1 + \cos(t+\theta) \tag{41}$$

with period 2π. In this case, periodic solutions to this dynamical system are of the form

$$x_1(t) = \frac{(k-1)\cos(t+\theta) + \sin(t+\theta)}{(k-1)^2+1},\ x_2(t) = \frac{\cos(t+\theta) + (1-k)\sin(t+\theta)}{(k-1)^2+1}. \tag{42}$$

Consequently, stationary points occur wherever

$$\frac{2(1-k)\cos\theta + k(k-2)\sin\theta}{\left((k-1)^2+1\right)^2} = \frac{(1-k)\cos\theta - \sin\theta}{(k-1)^2+1} = 0, \tag{43}$$

i.e., for $k=1$ and $\theta = n\pi$ for any integer n.

Alternatively, consider the Lagrangian

$$\begin{aligned} L\left(x(t),k,\theta,\mu_k,\mu_\theta,\mu_v,\ell_{\text{ode}}(t),\ell_{\text{bc}},\eta_k,\eta_\theta,\eta_v\right) &= \mu_v + \int_0^{2\pi} \ell_{\text{ode},1}(t)\big(\dot{x}_1(t) - x_2(t)\big)\,\mathrm{d}t \\ &+ \int_0^{2\pi} \ell_{\text{ode},2}(t)\big(\dot{x}_2(t) + x_2(t) + kx_1(t) - \cos(t+\theta)\big)\,\mathrm{d}t + \ell_{\text{bc},1}\big(x_1(0) - x_1(2\pi)\big) \\ &+ \ell_{\text{bc},2}\big(x_2(0) - x_2(2\pi)\big) + \eta_k\left(k-\mu_k\right) + \eta_\theta\left(\theta-\mu_\theta\right) + \eta_v\left(x_2(0)-\mu_v\right) \end{aligned} \tag{44}$$

in terms of the continuation parameters μ_k, μ_θ, and μ_v, and the Lagrange multipliers ℓ_{ode}, ℓ_{bc}, η_k, η_θ, and η_v. Necessary conditions for stationary points along the constraint manifold correspond to points $(x(t),k,\theta,\mu_k,\mu_\theta,\mu_v,\ell_{\text{ode}}(t),\ell_{\text{bc}},\eta_k,\eta_\theta,\eta_v)$ for which $\delta L = 0$ for any infinitesimal variations $\delta x(t)$, δk, $\delta\theta$, $\delta\mu_k$, $\delta\mu_\theta$, $\delta\mu_v$, $\delta\ell_{\text{ode}}(t)$, $\delta\ell_{\text{bc}}$, $\delta\eta_k$, $\delta\eta_\theta$, and $\delta\eta_v$. In this

case, these conditions take the form

$$\dot{x}_1 = x_2,\ \dot{x}_2 = -x_2 - kx_1 + \cos(t+\theta),\ x_1(0) = x_1(2\pi),\ x_2(0) = x_2(2\pi), \tag{45}$$

$$k - \mu_k = 0,\ \theta - \mu_\theta = 0, x_2(0) - \mu_v = 0, \tag{46}$$

$$-\dot{\ell}_{\mathrm{ode},1} + k\ell_{\mathrm{ode},2} = 0,\ -\dot{\ell}_{\mathrm{ode},2} - \ell_{\mathrm{ode},1} + \ell_{\mathrm{ode},2} = 0, \tag{47}$$

$$\ell_{\mathrm{ode},1}(2\pi) - \ell_{\mathrm{bc},1} = 0,\ -\ell_{\mathrm{ode},1}(0) + \ell_{\mathrm{bc},1} = 0, \tag{48}$$

$$\ell_{\mathrm{ode},2}(2\pi) - \ell_{\mathrm{bc},2} = 0,\ -\ell_{\mathrm{ode},2}(0) + \ell_{\mathrm{bc},2} + \eta_v = 0, \tag{49}$$

$$\int_0^{2\pi} \ell_{\mathrm{ode},2}(t)x_1(t)\,\mathrm{d}t + \eta_k = 0,\ \int_0^{2\pi} \ell_{\mathrm{ode},2}(t)\sin(t+\theta)\,\mathrm{d}t + \eta_\theta = 0, \tag{50}$$

$1 - \eta_v = 0$, and $\eta_k = \eta_\theta = 0$. Solutions to these conditions are given by

$$x_1(t) = (-1)^n \sin t,\ x_2(t) = (-1)^n \cos t,\ k = \mu_k = 1,\ \theta = \mu_\theta = n\pi,\ \mu_v = (-1)^n, \tag{51}$$

$$\ell_{\mathrm{ode},1}(t) = e^{t/2-\pi}\frac{e^\pi \sin\left(\frac{\sqrt{3}}{2}(2\pi - t)\right) + \sin\frac{\sqrt{3}t}{2}}{\sqrt{3}(\cosh\pi - \cos\sqrt{3}\pi)},\ \ell_{\mathrm{bc},1} = \frac{\sin\sqrt{3}\pi}{\sqrt{3}(\cosh\pi - \cos\sqrt{3}\pi)}, \tag{52}$$

$$\ell_{\mathrm{ode},2}(t) = e^{\frac{t}{2}-\pi}\frac{3\cos\frac{\sqrt{3}t}{2} - 3e^\pi\cos\left(\frac{\sqrt{3}}{2}(2\pi - t)\right) + \sqrt{3}\left(e^\pi\sin\left(\frac{\sqrt{3}}{2}(2\pi - t)\right) + \sin\frac{\sqrt{3}t}{2}\right)}{6(\cosh\pi - \cos\sqrt{3}\pi)}, \tag{53}$$

$$\ell_{\mathrm{bc},2} = \frac{3\cos\sqrt{3}\pi + \sqrt{3}\sin\sqrt{3}\pi - 3e^\pi}{6(\cosh\pi - \cos\sqrt{3}\pi)},\ \eta_k = \eta_\theta = 0,\ \eta_v = 1. \tag{54}$$

Stationary points along the solution manifold may be located using a method of successive continuation applied to the extended continuation problem obtained by combining (45)-(50) with $\eta_k - \nu_k = 0$, $\eta_\theta - \nu_\theta = 0$, and $\eta_v - \nu_v = 0$ in terms of the continuation variables $(x(t), k, \theta, \ell_{\mathrm{ode}}(t), \ell_{\mathrm{bc}}, \eta_k, \eta_\theta, \eta_v)$ and continuation parameters $(\mu_k, \mu_\theta, \mu_v, \nu_k, \nu_\theta, \nu_v)$. The dimensional deficit of this extended continuation problem equals 3. We get one-dimensional solution manifolds by designating two of the continuation parameters as inactive.

To simplify the analysis, we first eliminate the continuation variables $x(t)$, k, θ, $\ell_{\mathrm{ode}}(t)$, ℓ_{bc}, η_k, and η_θ from (45)-(50) and obtain

$$x_1(t) = \frac{(\mu_k - 1)\cos(t+\mu_\theta) + \sin(t+\mu_\theta)}{(\mu_k-1)^2 + 1},\ x_2(t) = \frac{\cos(t+\mu_\theta) + (1-\mu_k)\sin(t+\mu_\theta)}{(\mu_k-1)^2+1}, \tag{55}$$

$$k = \mu_k,\ \theta = \mu_\theta,\ \ell_{\mathrm{bc},1} = \frac{\eta_v\mu_k\sin\tilde{\mu}_k\pi}{\tilde{\mu}_k(\cosh\pi - \cos\tilde{\mu}_k\pi)},\ \ell_{\mathrm{bc},2} = \left(\frac{\sin\tilde{\mu}_k\pi - \tilde{\mu}_k\sinh\pi}{\tilde{\mu}_k(\cosh\pi - \cos\tilde{\mu}_k\pi)} - 1\right)\frac{\eta_v}{2}, \tag{56}$$

$$\eta_k = -\frac{2(1-\mu_k)\cos\mu_\theta + \mu_k(\mu_k-2)\sin\mu_\theta}{((\mu_k-1)^2+1)^2}\eta_v,\ \eta_\theta = -\frac{(1-\mu_k)\cos\mu_\theta - \sin\mu_\theta}{(\mu_k-1)^2+1}\eta_v, \tag{57}$$

$$\ell_{\mathrm{ode},1}(t) = e^{t/2-\pi}\frac{\mu_k\eta_v\left(\sin\frac{\tilde{\mu}_k t}{2} + e^\pi\sin\frac{\tilde{\mu}_k}{2}(2\pi - t)\right)}{\tilde{\mu}_k(\cosh\pi - \cos\tilde{\mu}_k\pi)}, \tag{58}$$

and

$$\ell_{\text{ode},2}(t) = e^{t/2-\pi}\frac{\eta_v\left(\tilde{\mu}_k\cos\frac{\tilde{\mu}_k t}{2} + \sin\frac{\tilde{\mu}_k t}{2} + e^{\pi}\left(\sin\frac{\tilde{\mu}_k}{2}(2\pi - t) - \tilde{\mu}_k\cos\frac{\tilde{\mu}_k}{2}(2\pi - t)\right)\right)}{2\tilde{\mu}_k\left(\cosh\pi - \cos\tilde{\mu}_k\pi\right)}, \tag{59}$$

where $\tilde{\mu}_k = \sqrt{4\mu_k - 1}$. The remaining conditions then imply that

$$\frac{\cos\mu_\theta + (1-\mu_k)\sin\mu_\theta}{(\mu_k-1)^2+1} - \mu_v = 0, \tag{60}$$

$$-\frac{2(1-\mu_k)\cos\mu_\theta + \mu_k(\mu_k-2)\sin\mu_\theta}{((\mu_k-1)^2+1)^2}\eta_v - \nu_k = 0, \tag{61}$$

$$-\frac{(1-\mu_k)\cos\mu_\theta - \sin\mu_\theta}{(\mu_k-1)^2+1}\eta_v - \nu_\theta = 0 \tag{62}$$

$$\eta_v - \nu_v = 0. \tag{63}$$

Now suppose that μ_v, μ_k, ν_v, and ν_θ are active and μ_θ and ν_k are inactive. Then, if $\nu_k = 0$, solutions to the corresponding restricted continuation problem are located on one of the three one-dimensional manifolds

$$\mu_v = \frac{\cos\mu_\theta + (1-\mu_k)\sin\mu_\theta}{(\mu_k-1)^2+1},\ \eta_v = \nu_v = \nu_\theta = 0, \tag{64}$$

$$\mu_v = \cos^2\frac{\mu_\theta}{2},\ \mu_k = 1 - \tan\frac{\mu_\theta}{2},\ \eta_v = \nu_v,\ \nu_\theta = \frac{\nu_v}{2}\sin\mu_\theta, \tag{65}$$

or

$$\mu_v = -\sin^2\frac{\mu_\theta}{2},\ \mu_k = 1 + \cot\frac{\mu_\theta}{2},\ \eta_v = \nu_v,\ \nu_\theta = \frac{\nu_v}{2}\sin\mu_\theta. \tag{66}$$

The manifold in (64) intersects the manifolds in (65) and (66) at the points

$$\mu_v = \cos^2\frac{\mu_\theta}{2},\ \mu_k = 1 - \tan\frac{\mu_\theta}{2},\ \eta_v = \nu_v = \nu_\theta = 0, \tag{67}$$

and

$$\mu_v = -\sin^2\frac{\mu_\theta}{2},\ \mu_k = 1 + \cot\frac{\mu_\theta}{2},\ \eta_v = \nu_v = \nu_\theta = 0, \tag{68}$$

respectively, corresponding to local extrema in the value of μ_v along the first manifold.

Notably, there is a unique point on each of the latter manifolds where $\eta_v = 1$. If we consider the restricted continuation problem obtained with μ_v, μ_k, μ_θ, and ν_θ active and ν_k and ν_v inactive and equal to 0 and 1, respectively, then solutions are located on the one-dimensional manifolds

$$\mu_v = \cos^2\frac{\mu_\theta}{2},\ \mu_k = 1 - \tan\frac{\mu_\theta}{2},\ \eta_v = 1,\ \nu_\theta = \frac{1}{2}\sin\mu_\theta, \tag{69}$$

and

$$\mu_v = -\sin^2\frac{\mu_\theta}{2},\ \mu_k = 1 + \cot\frac{\mu_\theta}{2},\ \eta_v = 1,\ \nu_\theta = \frac{1}{2}\sin\mu_\theta. \tag{70}$$

Notably, the points with $\mu_\theta = 2n\pi$ on the first manifold and $\mu_\theta = (2n+1)\pi$ on the second manifold, for any integer n, coincide with the stationary points found previously.

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

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

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

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

end


function J = linode_dx(t, x, p)

k = p(1,:);

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

end


function J = linode_dp(t, x, p)

x1 = x(1,:);
th = p(2,:);

J = zeros(2,2,numel(t));
J(2,1,:) = -x1;
J(2,2,:) = -sin(t+th);

end


function J = linode_dt(t, x, p)

th = p(2,:);

J = zeros(2,numel(t));
J(2,:) = -sin(t+th);

end


function dJ = linode_dxdx(t, x, p)
dJ = zeros(2,2,2,numel(t));
end
```

```
function dJ = linode_dxdp(t, x, p)

dJ = zeros(2,2,2,numel(t));
dJ(2,1,1,:) = -1;

end


function dJ = linode_dpdp(t, x, p)

th = p(2,:);

dJ = zeros(2,2,2,numel(t));
dJ(2,2,2,:) = -cos(t+th);

end


function dJ = linode_dtdx(t, x, p)
dJ = zeros(2,2,numel(t));
end


function dJ = linode_dtdp(t, x, p)

th = p(2,:);

dJ = zeros(2,2,numel(t));
dJ(2,2,:) = -cos(t+th);

end


function dJ = linode_dtdt(t, x, p)

th = p(2,:);

dJ = zeros(2,numel(t));
dJ(2,:) = -cos(t+th);

end
```

In the first stage of construction, we use the `ode_isol2coll` toolbox constructor to encode the trajectory constraint, as shown in the sequence of commands below.

```
>> prob = coco_prob;
>> prob = coco_set(prob, 'ode', 'autonomous', false);
>> [t0, x0]  = ode45(@(t,x) linode(t, x, [0.98; 0.3]), [0 2*pi], ...
     [0.276303; 0.960863]);
>> coll_args = {@linode, @linode_dx, @linode_dp, @linode_dt, ...
     @linode_dxdx, @linode_dxdp, @linode_dpdp, @linode_dtdx, ...
     @linode_dtdp, @linode_dtdt, t0, x0, {'k' 'th'}, [0.98; 0.3]};
>> prob1 = ode_isol2coll(prob, '', coll_args{:});
```

Here `'k'` and `'th'` represent the continuation parameters μ_k and μ_θ, respectively. We proceed to append boundary conditions defined in terms of the COCO-compatible function encodings

shown below.

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

x0 = u(1:2);
x1 = u(3:4);
T0 = u(5);
T  = u(6);

y = [x1(1:2)-x0(1:2); T0; T-2*pi];

end


function [data, J] = linode_bc_du(prob, data, u)
J = [-1 0 1 0 0 0; 0 -1 0 1 0 0; 0 0 0 0 1 0; 0 0 0 0 0 1];
end


function [data, dJ] = linode_bc_dudu(prob, data, u)
dJ = zeros(4,6,6);
end
```

As shown in the commands below, we associate `'v'` with the continuation parameter μ_v.

```
>> [data, uidx] = coco_get_func_data(prob1, 'coll', 'data', 'uidx');
>> maps = data.coll_seg.maps;
>> bc_funcs = {@linode_bc, @linode_bc_du, @linode_bc_dudu};
>> prob1 = coco_add_func(prob1, 'po', bc_funcs{:}, [], 'zero', 'uidx', ...
     uidx([maps.x0_idx; maps.x1_idx; maps.T0_idx; maps.T_idx]));
>> prob1 = coco_add_pars(prob1, 'vel', uidx(maps.x0_idx(2)), 'v');
```

The contributions to the adjoint equations corresponding to the zero and monitor functions associated with the single instance of the `'coll'` toolbox are appended to the continuation problem using the `adjt_isol2coll` constructor, as shown below.

```
>> prob1 = adjt_isol2coll(prob1, '');
```

This call initializes all Lagrange multipliers at 0 and introduces the continuation parameters `'d.k'` and `'d.th'` corresponding to ν_k and ν_θ, respectively. The following sequence of commands append the contributions to the adjoint equations corresponding to the boundary conditions and definition of the `'v'` continuation parameter.

```
>> [data, axidx] = coco_get_adjt_data(prob1, 'coll', 'data', 'axidx');
>> opt = data.coll_opt;
>> prob1 = coco_add_adjt(prob1, 'po', 'aidx', ...
     axidx([opt.x0_idx; opt.x1_idx; opt.T0_idx; opt.T_idx]));
>> prob1 = coco_add_adjt(prob1, 'vel', 'd.v', 'aidx', axidx(opt.x0_idx(2)));
```

We use the `'aidx'` flag in each call to `coco_add_adjt` to identify equations associated with variations in some subset of the continuation variables.

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

```
>> coco(prob1, 'linode1', [], 1, {'v' 'k' 'd.v' 'd.th'}, [0.9 2]);
```

We continue along a secondary branch through the branch point found in the first run by reconstructing the augmented continuation problem using the following sequence of commands.

```
>> bd1   = coco_bd_read('linode1');
>> BPlab = coco_bd_labs(bd1, 'BP');
>> prob2 = ode_BP2coll(prob, '', 'linode1', BPlab(1));
>> [data, uidx] = coco_get_func_data(prob2, 'coll', 'data', 'uidx');
>> maps  = data.coll_seg.maps;
>> prob2 = coco_add_func(prob2, 'po', bc_funcs{:}, data, 'zero', 'uidx', ...
     uidx([maps.x0_idx; maps.x1_idx; maps.T0_idx; maps.T_idx]));
>> prob2 = coco_add_pars(prob2, 'vel', uidx(maps.x0_idx(2)), 'v');
>> chart = coco_read_solution('linode1', BPlab(1), 'chart');
>> cdata = coco_get_chart_data(chart, 'lsol');
>> prob2 = adjt_BP2coll(prob2, '', 'linode1', BPlab(1));
>> [chart, lidx] = coco_read_adjoint('po', 'linode1', BPlab(1), ...
     'chart', 'lidx');
>> [data, axidx] = coco_get_adjt_data(prob2, 'coll', 'data', 'axidx');
>> opt   = data.coll_opt;
>> prob2   = coco_add_adjt(prob2, 'po', 'aidx', ...
     axidx([opt.x0_idx; opt.x1_idx; opt.T0_idx; opt.T_idx]), ...
     'l0', chart.x, 'tl0', cdata.v(lidx));
>> [chart, lidx] = coco_read_adjoint('vel', 'linode1', BPlab(1), ...
     'chart', 'lidx');
>> prob2 = coco_add_adjt(prob2, 'vel', 'd.v', 'aidx', ...
     axidx(opt.x0_idx(2)), 'l0', chart.x, 'tl0', cdata.v(lidx));
>> coco(prob2, 'linode2', [], 1, {'d.v', 'v', 'k' 'd.th'}, {[0 1], [.9 2]});
```

We use the `coco_get_chart_data` utility to extract a candidate tangent vector that is perpendicular to the primary solution branch and in the plane spanned by the tangent vectors to the primary and secondary solution branches. We use the `coco_read_adjoint` utility to extract initial values for the Lagrange multipliers from the stored solution at the branch point, as well as the corresponding integer indices in the overall vector of continuation variables.

The third, and final, stage of continuation results from the sequence of commands shown below.

```
>> bd2 = coco_bd_read('linode2');
>> lab = coco_bd_labs(bd2, 'EP');
>> prob3 = ode_coll2coll(prob, '', 'linode2', lab(2));
>> [data, uidx] = coco_get_func_data(prob3, 'coll', 'data', 'uidx');
>> maps = data.coll_seg.maps;
>> prob3 = coco_add_func(prob3, 'po', bc_funcs{:}, data, 'zero', 'uidx', ...
     uidx([maps.x0_idx; maps.x1_idx; maps.T0_idx; maps.T_idx]));
>> prob3 = coco_add_pars(prob3, 'vel', uidx(maps.x0_idx(2)), 'v');
>> prob3 = adjt_coll2coll(prob3, '', 'linode2', lab(2));
>>  chart = coco_read_adjoint('po', 'linode2', lab(2), 'chart');
>> [data, axidx] = coco_get_adjt_data(prob3, 'coll', 'data', 'axidx');
>> opt = data.coll_opt;
>> prob3   = coco_add_adjt(prob3, 'po', 'aidx', ...
     axidx([opt.x0_idx; opt.x1_idx; opt.T0_idx; opt.T_idx]), 'l0', chart.x);
>> chart = coco_read_adjoint('vel', 'linode2', lab(2), 'chart');
```

```
>> prob3 = coco_add_adjt(prob3, 'vel', 'd.v', 'aidx', ...
     axidx(opt.x0_idx(2)), 'l0', chart.x);
>> prob3 = coco_add_event(prob3, 'OPT', 'd.th', 0);
>> coco(prob3, 'linode3', [], 1, {'d.th' 'v' 'k' 'th'}, {[], [.9 2]});
```

Here, every special point with `'d.th'` equal to 0 that is detected during continuation is assigned the `'OPT'` label.

Exercises

1. Consider an autonomous encoding of the vector field in this section, by augmenting the state x with the phase of the excitation, and perform the appropriate theoretical and computational analysis.

2. Repeat the analysis in the previous exercise using a `'bvp'`-compatible encoding of the boundary conditions.

3. Use two coupled `'bvp'` instances to represent two periodic solutions of the harmonically excited linear oscillator

   $$\dot{x}_1 = x_2,\ \dot{x}_2 = -x_1 - x_2 + \cos x_3,\ \dot{x}_3 = \omega$$

   with values of ω that differ by some small but positive number ϵ. Use the adjoint necessary conditions to locate local extrema in the difference between their maximal values of x_1 under variations in ω and branch switch to drive the continuation parameter corresponding to the associated Lagrange multiplier to 1.

4. Consider the Lagrangian

   $$\begin{aligned} &L\left(x(t), \zeta, \omega, \alpha, A, \mu_a, \ell_{\text{ode}}(t), \ell_{\text{bc}}, \eta_\zeta, \eta_\omega, \eta_\alpha, \eta_a\right) = \\ &\quad \mu_A + \int_0^{2\pi/\omega} \ell_{\text{ode}}(t)^\top \left(\dot{x}(t) - f(x(t), \zeta, \omega, \alpha, A)\right) \mathrm{d}t + \ell_{\text{bc}}^\top f_{\text{bc}}(x(2\pi/\omega), x(0)) \\ &\qquad + \eta_\zeta\left(\zeta - \mu_\zeta\right) + \eta_\omega\left(\omega - \mu_\omega\right) + \eta_\alpha\left(\alpha - \mu_\alpha\right) + \eta_a\left(A - \mu_A\right), \end{aligned}$$

   where

   $$f(x, \zeta, \omega, \alpha, A) = \begin{pmatrix} x_2 \\ -2\zeta x_2 - x_1 - \alpha x_1^3 - A\cos x_3 \\ \omega \end{pmatrix},$$

   $$f_{\text{bc}}(x, y) = \begin{pmatrix} x_1 - y_1 \\ x_2 - y_2 \\ x_3 - y_3 - 2\pi \end{pmatrix},$$

   and μ_ζ, μ_α, and μ_ω are fixed. Show that $\mu_A = 2\alpha\eta_\alpha$ at a local extremum of L. Verify this result with COCO using two stages of continuation of a suitably constructed extended continuation problem.

---

9 Toolbox reference

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

9.1 Zero problems

For continuation of general trajectory segments, the zero problem is given in terms of the vector of continuation variables $u = (\upsilon_{bp}, T_0, T, p)$ by $\Phi(u) = 0$, where the column matrix υ_{bp} contains the unknown values of the state variables on the mesh of base points and

$$\Phi : u \mapsto \begin{pmatrix} \frac{T}{2N}\ \mathfrak{vec}\left(\kappa_F * F\left(T_0 + Tt_{cn}, \mathfrak{vec}_n\left(W \cdot \upsilon_{bp}\right), 1_{1,Nm} \otimes p\right)\right) - W' \cdot \upsilon_{bp} \\ Q \cdot \upsilon_{bp} \end{pmatrix} \tag{71}$$

is the corresponding family of zero functions. Here, t_{cn} denotes a set of collocation nodes on the interval $[0, 1]$. For an autonomous vector field, the continuation variable T_0 is associated with an inactive continuation parameter `'OID.coll.T0'`, where `OID` denotes an object instance identifier (the period is omitted when `OID` equals the empty string). The dimensional deficit then equals $n + q + 1$, where n is the number of state variables and q is the number of problem parameters. For a non-autonomous vector field, the dimensional deficit equals $n + q + 2$.

The discretization associated with the zero problem is represented by the two-dimensional array κ_F and the matrices W, W', and Q. Adaptive remeshing of the discretization involves changes to κ_F as well as, in the case of changes to the number of discretization intervals, to the matrices W, W', and Q.

In the current implementation of the `'coll'` toolbox, the zero problem for simultaneous continuation of trajectory segments and solutions to the corresponding variational equation is given by appending the one-dimensional array $\mathfrak{vec}\left(\Delta_{bp}\right)$ to the continuation variables and the one-dimensional array

$$\mathfrak{vec}\begin{pmatrix} \frac{T}{2N}\mathfrak{diag}\big(\kappa_{\partial_x F} * \partial_x F\left(T_0 + Tt_{cn}, \mathfrak{vec}_n\left(W \cdot \upsilon_{bp}\right), 1_{1,Nm} \otimes p\right)\big) \cdot W \cdot \Delta_{bp} - W' \cdot \Delta_{bp} \\ Q \cdot \Delta_{bp} \end{pmatrix} \tag{72}$$

to the corresponding family of zero functions. Each column of Δ_{bp} contains the unknown values of a solution to the variational equation on the mesh of base points. For an autonomous vector field, the total dimensional deficit now equals $n(m+1)+q+1$, where m is the number of simultaneous copies of the variational equation, i.e., the number of columns of Δ_{bp}. For a non-autonomous vector field, the total dimensional deficit equal $n(m + 1) + q + 2$.

In addition to κ_F, W, W', and Q, the discretization of the problem of simultaneous continuation of a trajectory segment and solutions to the corresponding variational problem is represented by the three-dimensional array $\kappa_{\partial_x F}$. Adaptive remeshing of the discretization involves changes to $\kappa_{\partial_x F}$ that are consistent with changes made to κ_F.

In the current implementation of the `'coll'` toolbox, the zero problem for multi-segment boundary-value problems includes multiple instances of the appropriate zero problem for a trajectory segment, as well as the imposition of boundary conditions that depend on the collection of interval lengths and trajectory end points, as well as the problem parameters. Specifically, if T_0, T, $\upsilon_{bp,0}$, $\upsilon_{bp,1}$, and p denote arrays of the corresponding elements of the collection of continuation variables, then the additional zero functions are of the form

$$(T_0, T, \upsilon_{bp,0}, \upsilon_{bp,1}, p) \mapsto f_{bc}\,(T_0, T, \upsilon_{bp,0}, \upsilon_{bp,1}, p) \tag{73}$$

for a non-autonomous problem, and

$$(T, \upsilon_{bp,0}, \upsilon_{bp,1}, p) \mapsto f_{bc}\,(T, \upsilon_{bp,0}, \upsilon_{bp,1}, p) \tag{74}$$

for an autonomous problem, in terms of some function f_{bc}.

9.2 Calling syntax

The calling syntax for 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_isol2coll` toolbox constructor, the `varargin` input argument adheres to the following syntax:

```
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 a 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. Notably, if adjoint equations are to be constructed using the `adjt_isol2coll` constructor, then the preceding call to `ode_isol2coll` must include explicit function handles to encodings of the Jacobians with respect to x, p, and (as appropriate) t, 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. 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.

In the case of the ode_isol2bvp constructor, the varargin input argument adheres to the syntax

```
(coll | {{coll} ...}) [pnames] @bc [@dbcdx] [bc_data [@bc_update]] [opts]
```

where coll indicates an argument that matches the varargin input syntax for ode_isol2coll for a single trajectory segment, but omits the pnames argument, as the latter must be common to all the segments. An error is thrown if string labels are included, or if the number of problem parameters are not the same for all segments. An optional designation of string labels for continuation parameters assigned to track the problem parameters is provided with pnames, which is either a single string or a cell array of strings. An error is thrown if the number of string labels in this optional argument, when present, differs from the number of elements of p0. The @bc, @dbcdx, and @bc_update input arguments denote required and optional function handles to encodings of the boundary condition function f_{bc}, its Jacobian with respect to the input arguments (in the non-autonomous case, T_0) T, $v_{bp,0}$, $v_{bp,1}$, and p, and the function used to update the data input argument of these encodings, respectively. Initial content for data is assigned in the bc_data input argument.

For each of the ode_coll2coll, ode_BP2coll, ode_bvp2bvp, and ode_BP2bvp toolbox constructors, the varargin input argument adheres to the syntax

```
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 the ode_isol2coll, ode_coll2coll, and ode_BP2coll constructors, the optional opts argument may equal either of the strings '-coll-end' or '-end-coll'. Similarly, for the ode_isol2bvp, ode_bvp2bvp, and ode_BP2bvp constructors, the optional opts argument may equal either of the strings '-bvp-end' or '-end-bvp'. In either case, this denotes explicitly the end of the sequence of arguments to a 'coll' toolbox constructor. For ode_coll2coll and ode_bvp2bvp, 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_isol2coll, ode_coll2coll, and ode_BP2coll 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_bvp2bvp, 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.

9.3 Adjoint functions

For continuation of general trajectory segments, 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_isol2coll` constructor, following a preceding call to the `ode_isol2coll` constructor that includes function handles to explicit encodings of these Jacobians. Specifically, in the call

```
prob = adjt_isol2coll(prob, oid)
```

the `oid` argument denotes an object identifier associated with the toolbox instance created by the preceding call to `ode_isol2coll`. The corresponding components of λ are initialized to 0.

If the preceding call to `ode_isol2coll` includes an explicit list of parameter labels, then the corresponding additions to the adjoint equations are automatically encoded by the call to `adjt_isol2coll`. The corresponding components of the vector of continuation multipliers η are initialized to 0.

In a similar fashion, a call to `ode_coll2coll` or `ode_BP2coll` may be followed by a call to `adjt_coll2coll` or `adjt_BP2coll`, respectively, with identical arguments, in order to append the contributions to the adjoint equations associated with the reconstructed continuation problem. In either case, the associated elements of the vectors of continuation multipliers λ and η are automatically initialized from the corresponding values stored in a solution file.

For continuation of families of constrained trajectory segments, the contributions to the adjoint equations associated with variations in υ_{bp}, T_0, T, and p are expressed in terms of the Jacobians $\partial_t F(t,x,p)$, $\partial_x F(t,x,p)$, $\partial_p F(t,x,p)$, and ∂f_{bc}, 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_isol2bvp` constructor, following a preceding call to the `ode_isol2bvp` constructor that includes function handles to explicit encodings of these Jacobians. Specifically, in the call

```
prob = adjt_isol2bvp(prob, oid)
```

the `oid` argument denotes an object identifier associated with the toolbox instance created by the preceding call to `ode_isol2bvp`. The corresponding components of λ are initialized to 0.

If the preceding call to `ode_isol2bvp` includes an explicit list of parameter labels, then the corresponding additions to the adjoint equations are automatically encoded by the call to `adjt_isol2bvp`. The corresponding components of the vector of continuation multipliers η are initialized to 0.

In a similar fashion, a call to `ode_bvp2bvp` or `ode_BP2bvp` may be followed by a call to `adjt_bvp2bvp` or `adjt_BP2bvp`, respectively, with identical arguments, in order to append the contributions to the adjoint equations associated with the reconstructed continuation problem. In either case, the associated elements of the vectors of continuation multipliers λ

and η are automatically initialized from the corresponding values stored in a solution file.

9.4 Continuation parameters

The inclusion of the `pnames` optional argument in the call to the `ode_isol2coll` toolbox constructor 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 `ode_coll2coll` or `ode_BP2coll`. The same holds for a call to the `ode_isol2bvp` constructor and when restarting continuation using either `ode_bvp2bvp` or `ode_BP2bvp`. A subsequent call to `adjt_isol2coll`, `adjt_coll2coll`, `adjt_BP2coll`, `adjt_isol2bvp`, `adjt_bvp2bvp`, or `adjt_BP2bvp` 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.

All the `'coll'` toolbox constructors encode two nonembedded continuation parameters `'OID.coll.err'` and `'OID.coll.err_TF'` for each trajectory segment and corresponding object instance identifier `OID`. These are associated with an estimate of the discretization error and the ratio between this error and an error tolerance. The detection of a special point denoted by `'MXCL'` is triggered when the second of these exceeds 1 unless the optional setting `'MXCL'` is set to to `false`.

For an autonomous vector field, an embedded continuation parameter `'d.OID.coll.T0'` is encoded in the continuation problem structure and designated as active. This parameter corresponds to an element of the vector of continuation multipliers η that is initialized to 0.

When the optional setting `'var'` is set to `true`, the `ode_isol2coll` and `ode_coll2coll` constructors encode a nonembedded monitor function whose output is empty and, therefore, not associated with a continuation parameter. This monitor function stores a temporary copy of a nonsingular matrix solution to the variational equation for use by other monitor functions, for example to compute Floquet multipliers for a periodic orbit. For a multi-segment boundary-value problem, each trajectory segment is associated with a separate instance of the `'var'` toolbox setting. The `coco_set` core utility can be used to set these individually or collectively, as described in *Recipes for Continuation*.

9.5 Toolbox settings

Optional settings associated with 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) and the degree of the interpolating polynomials (`'NCOL'` with default value 4). It is also possible to assign a non-default value to the discretization error tolerance (`'TOL'`) used to trigger the `'MXCL'` special point, although it may be best to do so only by

changing the global COCO tolerance, since this will ensure that the error tolerance used for 'coll' is consistent with the value used by the nonlinear solver.

In the absence of adaptive remeshing, the number of discretization intervals remain unchanged during continuation. With an atlas algorithm that supports mesh adaptation, the frequency of adaptation is associated with the optional toolbox setting 'NAdapt' of the 'cont' toolbox. The default value for this setting is 0, corresponding to no adaptation. Advanced settings of the 'coll' toolbox may be used to control the interval of allowable integer values ('NTSTMN' and 'NTSTMX' with default values min(5,'NTST') and max(100,'NTST'), respectively) for the number of discretization intervals in the presence of adaptive remeshing. Adaptive changes to the mesh occur only if the estimated discretization error is outside of an adaptation window given by a lower bound ('TOLDEC' with default value 'TOL'/5) and an upper bound ('TOLINC' with default value 'TOL'/20).

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

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

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

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

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

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

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

9.6 Toolbox output

By definition, the bifurcation data cell array stored during continuation and returned by the coco entry-point function (given a receiving variable) includes columns with headers '||OID.x||_{L_2[0,T]}', '||OID.x||_{L_2[0,1]}', and 'OID.NTST' with data given by a quadrature-approximation of the $\mathcal{L}_2$ norm of the trajectory segment on the interval $[T_0, T_0 + T]$, the $\mathcal{L}_2$ norm of a time-rescaled version of the trajectory segment on the interval $[0, 1]$, and the number of mesh interval, respectively, and with OID representing an object instance identifier (the period is omitted when OID equals the empty string). All continuation parameters are included in the bifurcation data cell array by default, but printed to screen during continuation only if included in the list of arguments to the coco entry-point function.

For general trajectory segments, the sol output argument of the coll_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),
- the vector of continuation variables (in the `u` field),
- the tangent vector to the corresponding curve segment (in the `t` field).

For branch points (located by the atlas algorithm) the `t0` field contains a singular vector normal to `t`. In the case of simultaneous continuation of solutions to the variational problem, the field `var.v` contains the array of perturbations to the initial point on the trajectory segment.

For constrained families of trajectory segments, the `sol` output argument of the toolbox solution extractor `bvp_read_solution` contains a struct array, the i-th element of which includes

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

The `coll_plot_theme` and `bvp_plot_theme` toolbox utilities define the default visualization theme for the `'coll'` toolbox. The command

```
>> thm = coll_plot_theme('seg')
```

assigns the default theme for visualization of the results of continuation of general trajectory segments to the `thm` variable. Similarly, the command

```
>> thm = bvp_plot_theme('segs')
```

assigns the default theme for visualization of the results of continuation of constrained families of trajectory segments to the `thm` variable.

9.7 Developer's interface

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

an object instance identifier to the string 'bvp'. The object instance identifier of the 1st (2nd, 3rd, ...) instance of 'coll' embedded in an instance of 'bvp' is obtained by appending 'seg1' ('seg2', 'seg3', ...) to the 'bvp' toolbox instance identifier.

The coco_get_func_data core utility may be used to extract

- the toolbox data structure associated with a trajectory segment continuation problem (the 'data' option with function identifier 'OID.coll') or with a constrained boundary-value problem (the 'data' option with function identifier 'OID.bvp');
- the function dependency index set associated with the basic trajectory segment zero problem (the 'uidx' option with function identifier 'OID.coll'), with the variational zero problem (the 'uidx' option with function identifier 'OID.coll.var'), or with the boundary conditions zero problem of a constrained boundary-value problem (the 'uidx' option with function identifier 'OID.bvp').

For each trajectory segment zero problem, the toolbox data structure contains several fields associated with the 'ode' toolbox family. These include function handles to the corresponding vector field (fhan), to its Jacobians (dfdxhan, dfdphan, and dfdthan), and to functions evaluating the second derivatives with respect to the state variables, problem parameters, and time (dfdxdxhan, dfdxdphan, dfdpdphan, dfdtdxhan, dfdtdphan, and dfdtdthan), a cell array of string labels for the continuation parameters associated with problem parameters (pnames), the state-space dimension (xdim), and the number of problem parameters (pdim).

As shown in the examples and described further in the documentation of the coll_add interface function, the content of the coll_seg field of the toolbox data structure associated with each trajectory segment continuation problem includes context-independent arrays of integer indices that reference the discretization of the state variables (maps.xbp_idx), the initial time (maps.T0_idx), the interval length (maps.T_idx), the problem parameters (maps.p_idx), the initial point on the trajectory segment (maps.x0_idx), and the final point on the trajectory segment (maps.x1_idx), respectively. In the case that the 'var' toolbox option is set to true, the content of the coll_tst.M field of the toolbox data structure includes a nonsingular solution to the variational equation obtained from a nonembedded monitor function.

The coco_get_adjt_data core utility may be used to extract the adjoint row (the 'afidx' option) and column (the 'axidx' option) index sets as well as the toolbox adjoint data structure (the 'data' option). The content of the coll_opt field of the adjoint data structure associated with a trajectory segment includes context-independent arrays of integer indices for the columns associated with collocation nodes (xcn_idx), initial (x0_idx) and final (x1_idx) end points, initial time (T0_idx) and interval length (T_idx), and problem parameters (p_idx), respectively.

For a trajectory segment continuation problem that contains the variational zero problem, the content of the coll_var field of the toolbox data structure includes context-independent arrays of integer indices that reference the part of the solution to the variational problem corresponding to the initial point on the trajectory segment (v0_idx) and the final point on the trajectory segment (v1_idx), respectively.

The toolbox data structure associated with a constrained boundary value problems includes fields specifying the number of trajectory segments (`nsegs`) and a cell array of toolbox instance identifiers for each of the trajectory segment zero problems (`cids`).

The toolbox data structure associated with the basic trajectory segment continuation problem or with a constrained boundary-value problem contains a number of implementation-dependent internal fields, whose use may change in the future. Accessing such internal fields is deprecated.