SUPPLEMENTARY MATERIALS: Isochron foliations and global bifurcations: A case study

James Hannam*, Bernd Krauskopf*, and Hinke M. Osinga*

SM1. Implementation of isochron computations within CoCo. This guide walks through the computation of two foliations of isochrons in region $\boldsymbol{D}$, the forward-time isochron foliation $\mathcal{I}(\Gamma^s)$ of the stable periodic orbit Γ^s and the backward-time isochron foliation $\mathcal{U}(q_+)$ of the equilibrium q_+, as depicted in subsection 3.5. We approach the computation of each isochron via the continuation of trajectory segements that solve suitable two-point boundary value problems (BVPs), specifically those introduced in [SM10] and illustrated in [SM7, Figure 2]. To our knowledge this BVP approach to the computation of isochrons has been performed exclusively with the collocation BVP solver and pseudo-arclength continuation routine of the package AUTO-07P [SM3]. The implementation we present here uses the MATLAB-based sofware CoCo [SM1, SM2], which offers a number of advantages. While it uses also collocation and pseudo-arclength continuation, it has been designed specifically for setting up and solving multi-segment BVPs, such as those required for computing isochron foliations. Namely, CoCo allows for the native continuation of solutions to BVPs contructed from multiple orbit segments, each with its own and separate mesh discretization. CoCo inherently shifts mesh points to adapt the different meshes during the continuation of solutions to a BVP; while it also uses mesh adaptation, AUTO-07P uses a single mesh common to all segments of the overall BVP. Moreover, CoCo provides the option to dynamically add or remove mesh points from the discretization during continuation. These features greatly improve the stability of computations of solution families of multisegment BVPs. The user has the option of setting both a residual and absolute tolerance for solutions in CoCo, whereas only residual tolerances are accessible to users of AUTO-07P. While both packages consider their tolerances as an aggregate over a trajectory segment as represented by collocation, this difference is significant because we compute isochrons while referencing only one point along an orbit segment.

An improvement of our implementation is that we make use of the concept of a fundamental domain (see subsection SM1.4.5) to avoid a discontinuity issue of trajectory segements used to solve the BVP for different multiples of the period of the periodic orbit. Here we make use of the feature that CoCo's design invites the implementation of new toolboxes, which allows us to implement the passing of tangent vectors to the continuation of the BVP to help ensure the smoothness of the computed isochrons. While it is also possible to achieve this in AUTO-07P in principle, we do not believe that any such implementation would be as straightforward or user-friendly as in CoCo. Finally, an advantage of a Matlab implementation is that the data is readily available for plotting or rendering.

*Department of Mathematics, The University of Auckland, Private Bag 92019, Auckland 1142, New Zealand (James.Hannam@auckland.ac.nz, B.Krauskopf@auckland.ac.nz, H.M.Osinga@auckland.ac.nz).

SM1.1. Overview of the BVPs required. The set up that we use involves a multi-step process that involves a number of BVPs. A computation of isochrons of a periodic orbit starts with an orbit segment $\boldsymbol{\gamma}(t)$ that represents it, which satisfies periodic boundary conditions with period T_Γ. A BVP is then set up for an orbit segment $\boldsymbol{x}(t)$ that defines the time-T_Γ map to a linear approximation of $I_\theta(\Gamma)$, given as a short line segment that acts as an (approximate) fundamental domain. We then compute the one-dimensional isochron $I_\theta(\Gamma)$ as the smooth manifold traced out by the start point $\boldsymbol{x}(0)$ during a continuation run, where the end point $\boldsymbol{x}(T_\Gamma)$ vary along the fundamental domain. We remark that, while both AUTO-07P and CoCo rescale time such that orbit segments are defined over the time interval $[0,1]$, CoCo hides this fact, allowing the user to implement the BVP in the natural time $t \in [0,T]$ of the original (nonscaled) system.

The first BVP facilitates the rotation of a periodic orbit Γ by a phase θ relative to some reference point $\gamma_0 \in \Gamma$, which is by convention chosen as the maximal point in the first coordinate of Γ. This is achieved by separating $\boldsymbol{\gamma}(t) = \Gamma$ into two segments, the phase segment

$$\boldsymbol{x_1} = \{\boldsymbol{\gamma}(t) \in \Gamma \,|\, t \in [0, \theta\, T_\Gamma]\},$$

and the anchoring segment

$$\boldsymbol{x_2} = \{\boldsymbol{\gamma}(t) \in \Gamma \,|\, t + \theta\, T_\Gamma \in [\theta\, T_\Gamma, T_\Gamma]\},$$

and then varying the phase θ to the desired phase θ^* such that $\gamma_{\theta^*} = \boldsymbol{x_1}\big(\theta^*\, T_\Gamma\big)$. The anchoring and phase segments are subject to continuity boundary conditions at each end, and we further require that $\mathbf{F}(\boldsymbol{x_1}(0)) \cdot e_1 = 0$, where $\mathbf{F}(\cdot)$ is the vector field and e_1 is the unit vector of the first coordinate of the system. The rotated periodic orbit $\boldsymbol{\gamma}^*(t) = \Gamma^*$ is, hence, constructed by appending $\boldsymbol{x_1}(t)$ to $\boldsymbol{x_2}(t)$ such that $\boldsymbol{\gamma}^*(0) = \gamma_{\theta^*}$.

The second BVP is set up in two stages. Firstly, CoCo's collocation toolbox natively implements the variational equations, and a simple method for computing the Floquet multipliers and bundles using this toolbox is presented in [SM2]. We supplement this BVP with that above to allow the rotation of both the periodic orbit and its Floquet vector at the base point. To this end, the Floquet bundle from the previous stage is also separated into phase and anchoring segments. We subject the Floquet bundle to constraints as presented in [SM6] at γ_θ, and periodicity conditions at γ_0. Then, as θ is varied during continuation, solutions for both the periodic orbit and the Floquet vector of each desired phase are saved for the subsequent computation of isochrons.

Next, we use the appropriately rotated periodic orbit as the first solution $\boldsymbol{x}(t)$ that satisfies a BVP that defines the time-T_Γ map to the linear approximation $\boldsymbol{w_\theta}$ of $I_\theta(\Gamma)$ as determined previously. From this BVP we first construct the (approximate) fundamental domain $\boldsymbol{s_\theta}$ for this isochron as given by the vector $\boldsymbol{s_\theta}$ between $\boldsymbol{x}(0)$ and $\boldsymbol{x}(T_\Gamma)$, defined by a specified maximum orthogonal distance $\delta_{\max}$ of $\boldsymbol{x}(0)$ from $\boldsymbol{w_\theta}$ (while $\boldsymbol{x}(T_\Gamma)$ lies along $\boldsymbol{w_\theta}$); the associated orbit segment $\boldsymbol{s}(t)$ is saved for later use. We then consider the time-kT_Γ map for increasing integer k with $\boldsymbol{x}(kT_\Gamma)$ along the vector $\boldsymbol{s}(t)$. This allows us to compute the isochron $I_\theta(\Gamma)$ as successive arcs $I^k_\theta(\Gamma)$, where each such arc is the smooth one-dimensional curve that is traced out by $\boldsymbol{x}(0)$ as $\boldsymbol{x}(T_\Gamma)$ varies during a continuation over the length of $\boldsymbol{s_\theta}$ under the time-kT_Γ map.

The arc number k is increased each time that $\boldsymbol{x}(kT_\Gamma)$ reaches the end of $\boldsymbol{s_\theta}$ by appending the orbit segment $\boldsymbol{s}(t)$ to $\boldsymbol{x}(t)$. The resulting orbit segment has its end point at the base of $\boldsymbol{s_\theta}$ and, hence, its continuation traces out the next arc for the $(k+1)T_\Gamma$-map. When a sufficient number of arcs has been computed, the respective branch of the isochron is simply given by $I_\theta(\Gamma) = \bigcup_{i=0}^{k} I_\theta^k(\Gamma)$. The other branch of $I_\theta(\Gamma)$ is found in the same way by starting the computation of the fundamental domain from the vector $-\boldsymbol{w_\theta}$.

SM1.2. Toolboxes as part of the installation. The required BVPs, along with two additional toolboxes to facilitate the computation of initial solutions, are implemented with CoCo's trajectory collocation (`'coll'`) and boundary value problem (`'bvp'`) toolboxes. Rotation of periodic orbits by a phase $\theta \in [0, 1)$ and rotation of periodic orbits to satisfy the zero-phase convention are achieved with the rotate phase point toolbox (`'rpp'`), which is covered in subsection SM1.4.3. The computation of Floquet multipliers of a periodic orbit, as well as their Floquet bundle via rotation of the periodic orbit, are achieved by the Floquet toolbox (`'flqt '`); this is covered in subsection SM1.4.4. The computation of arcs of an isochron is achieved by the isochron toolbox (`'iscrn'`), which is covered in subsection SM1.4.5 for attracting periodic orbits, and in subsection SM1.5 for repelling equilibria. These toolboxes were developed to compute isochrons in two-dimensional systems, and to provide a relatively user-friendly implementation for isochron computation.

The function files that make up these toolboxes are included as part of the supplementary materials with this paper, in a directory named `iscrn`. For temporary use, this folder may simply be placed in the searchable path of the active MATLAB directory. For long term use it is advised to add this folder to the CoCo install directory; see [SM1] for installation instructions. To add the 'iscrn' toolbox to MATLAB's path, open the file `startup.m` (which is added when installing CoCo) located at the path returned by the MATLAB command `userpath`, add the line `addpath(fullfile(PATH_TO_COCO, 'iscrn', 'toolbox'))`, where `PATH_TO_COCO` is a string containing the path to CoCo's install directory. There are a number of files included in the folder `iscrn/examples` that are intended for use by the reader when following along this guide. These should simply be placed in the search path of the active MATLAB directory. Specifically, the files `po_forward_demo.m` and `ep_backward_demo.m` in the `tube` subdirectory contain the code presented in this guide, in subsection SM1.4 and subsection SM1.5, respectively. Additionally, the file `isochron_foliation/isochron_foliation.m` is a script that programatically computes a foliation of isochrons, forward-time or backward-time, of periodic orbits or equilibria. The latter file is not directly discussed in this document, but may be understood from the embedded comments and the general guidance provided here. The use of the toolboxes within `iscrn` assume familiarity with CoCo, and the reader may find it useful to also refer to the technical manuals located in the `help` folder, located in CoCo's install directory; specifically, the 'Short Developer's Reference for CoCo' (`COCO_shortRef.pdf`) and the 'Trajectory Collocation Toolbox' (`COLL-Tutorial.pdf`).

We ask any reader who adapts and uses our CoCo toolboxes for their own research to include a reference to this supplementary material in any published work.

SM1.3. CoCo-comparable vector field functions. We encode (10) below as the function `tube` for use throughout this guide. This specific encoding is in the format of a CoCo-compatible function file for a vector field of a system of ODEs; it is in the vectorized format to

speed up computation, which requires that each of these variables has a `:` in the last dimension of their arrays. CoCo requires this function to return a single variable `Y` as the right hand side of a vector field; it takes the variables `u` and system parameters `p` as input arguments.

```
function Y = tube(u, p)
%Tube

x = u(1,:);
y = u(2,:);

a  = p(1,:);
b  = p(2,:);
c  = p(3,:);
d  = p(4,:);
mu = p(5,:);

r2 = x .^2 + y .^2;

Y(1,:) = mu .* a .* x -y - b .* x .* r2;
Y(2,:) = x + mu .* (a + c) .* y - (b + d) .* y .* r2;

end
```

When using CoCo for relatively difficult continuation problems, it is advised to supply the Jacobians of the vector field along with the vector field itself. CoCo is able to perform continuation without user-supplied Jacobian functions, but CoCo's default Jacobian approximations tends to be slower and less accurate. We encode the Jacobian with respect to the state variables as `tube_DFDX` and the Jacobian with respect to the system parameters as `tube_DFDP`, each each in the vectorized format.

SM1.4. Computing the isochrons of an attracting periodic orbit. This section covers the steps required to compute the isochron foliation of the attracting periodic orbit Γ^s in region $\boldsymbol{D}$; the required code is in the file `po_forward_demo.m` in `iscrn/toolbox/examples/tube`. Subsection SM1.4.1 describes the settings used throughout these computations and subsection SM1.4.2 describes how the periodic orbit Γ^s can be found via numerical integration. Then subsection SM1.4.3 covers the use of the `'rpp'` toolbox; here the reader is guided through computing Γ^s via numerical integration, followed by implementing a rotation of Γ^s to satisfy the zero-phase convention. Subsection SM1.4.4 covers the use of the `'flqt'` toolbox; here we first show how to compute a single Floquet vector and then an entire bundle of Floquet vectors for desired phases. Subsection SM1.4.5 describes the basic use of the `'iscrn'` toolbox; here we start with the computation of the fundamental domain and then guide the reader through the computation of subsequent arcs of a given isochron.

SM1.4.1. Computational Settings. It is advised that the reader keeps the accuracy settings similar throughout the various computations needed to compute the initial solution for the `'iscrn'` toolbox. To this end, we include below the function `orbit_settings`, which modifies the continuation problem structure `prob` by using the function `coco_set`, which is well documented in the 'Short Developer's Reference for CoCo' except for the options used in

line 8. For problems using the 'coll' toolbox, one should set computational tolerances by the 'all' option, instead of the options available under the 'coll' option.

```
function prob = orbit_settings( prob )

prob = coco_set( prob, 'ode', 'vectorized', 'on' );
prob = coco_set( prob, 'cont', 'ItMX', 1e6, 'NPR', 0, 'NAdapt', 1,...
         'h_min', 5e-10, 'h0', 5e-3, 'h_max', 5e-2 );
prob = coco_set( prob, 'coll', 'NTST', 50 );
% prob = coco_set( prob, 'coll', 'NTST', 100 );  % C_u
prob = coco_set( prob, 'all', 'TOL', 5e-8 );

end
```

To ensure that all arcs of $I_\theta(\Gamma^s)$ are computed to the same level of accuracy, we use the function `iscrn_settings` as below. This particular function is used for isochrons in each of regions $\boldsymbol{A}$–$\boldsymbol{F}$; certain isochrons require different settings and these are recorded in this file. The tolerances and step sizes indicated in `iscrn_settings` are set for high accuracy at the expense of computation time and file sizes. The changes to the 'ItMX' and 'NPR' settings of 'cont' instruct CoCo to only perform continuation in one (forward) direction and only save and print to screen special solutions during continuation, respectively. As with `orbit_settings`, we change the absolute tolerance using the 'all' option, rather altering the native 'coll' toolbox option.

```
function prob = iscrn_settings( prob )

prob = coco_set( prob, 'ode', 'vectorized', 'on' );
prob = coco_set( prob, 'cont', 'ItMX', [ 0, 1e6 ], 'NPR', 0, 'NAdapt', 1,...
         'h_min', 5e-10, 'h0', 5e-5, 'h_max', 5e-2 );
prob = coco_set( prob, 'coll', 'NTST', 50 );  % default
% prob = coco_set( prob, 'coll', 'NTST', 60 );  % B_u; C_u 0.4 0.5 0.7 0.9; D_eq 0.4
prob = coco_set( prob, 'all', 'TOL', 5e-9 );  % default
% prob = coco_set( prob, 'all', 'TOL', 5e-8 );  % B_u; C_u 0.4 0.5 0.7 0.9; D_eq 0.4
end
```

The choice of 'NTST' defines the number of mesh points that should be used in the initial solution that satisfies the isochron BVP. Since the subsequent arc $I_\theta^{k+1}(\Gamma^s)$ of the computed isochron is defined by the time-$(k+1)T_{\Gamma^s}$ map, the number of mesh points involved in the computation grows as k'NTST'. Naturally, this increases the memory and computation time required for the computation. As such, 'NTST' should be chosen in consideration of the total number of arcs required to compute the isochron, usually after a test computation. It is advised to use the 'coll' toolbox's mesh adaption option, which adds and removes mesh points during continuation, for computational stability.

SM1.4.2. Computing Γ^s. To begin, we need a numerical representation of Γ^s, which can be achieved in a number of ways for further use in CoCo. Typically when applying numerical continuation to systems of ordinary differential equations (ODE's) we may find a periodic orbit as a result of related bifurcations. However, as described in section 2, Γ^s arises from a saddle-node of periodic orbits and disappears in a heteroclinic bifurcation involving saddle

points. Neither the saddle-node nor the heteroclinic bifurcations serve as vehicles to compute Γ^s in CoCo, and instead we compute Γ^s by numerical integration.

CoCo-comparable functions for vector fields are designed for easy use with MATLAB's ODE integrators, such as `ode45`, and we may invoke the `tube` function *anonymously* with `ode45` to integrate (10) by prepending its call with `@(t, x)`.

```
>> p0       = [−0.66; 0.5; 2.5; 2.5; 1.0];
>> [t0, x0]  = ode45(@(t, x) tube(x, p0), [0, 100], [0; 0.25]);
```

Appropriate values for the system parameters of (10) are stored in `p0`, and we choose the integration time, `100`, to be sufficiently large for our choice of initial condition, `[0; 0.25]`. The trajectory computed with `ode45` accumulates onto Γ^s. From its time series we make a reasonable guess of the period T_{Γ^s} of Γ^s. Next, either take a segment at the end of `t0` and `x0` commensurate with the guess for T_{Γ^s}, or use `ode45` to integrate approximately one period from the end point of `x0` to get a numerical approximation of Γ^s for use with the `'rpp'` toolbox.

```
>> [t0, x0]  = ode45(@(t, x) tube(x, p0), [0, 10.5], x0(end,:)');
```

As long as the estimate of T_{Γ^s} is sufficiently accurate, `t0` and `x0` returned by `ode45` will converge to a solution that satisfies the periodic boundary conditions in `ode_isol2rpp`.

SM1.4.3. RPP: Rotating periodic orbits. The `'rpp'` toolbox is designed to facilitate the rotation of a periodic orbit Γ by a phase $\theta \in [0, 1)$ relative to some reference point $\gamma_0 \in \Gamma$. While the toolbox separates the supplied periodic orbits into two segments as described in subsection SM1.1, both the input to its constructor `ode_isol2rpp` and the output form its reader `rpp_read_solution` are single segment periodic orbits. This toolbox is capable of rotating a periodic orbit to satisfy the zero-phase convention, and is extended into the `'flqt'` toolbox to facilitate the rotation of a periodic orbit with its Floquet vector.

Applying the zero-phase convention. To make sure that the numerical representation `x0` of Γ^s satisfies the zero-phase convention, that is, the zero-phase point γ_0 is at the maximum in the first coordinate along Γ^s, we employ the `ode_isol2rpp` constructor function with the option `'−zpp'` set to `'zero_phase'`. The following commands construct a continuation problem to this effect.

```
>> p    = {'a', 'b', 'c', 'd', 'mu_s'};    % paramter names
>> prob = coco_prob();
>> prob = orbit_settings(prob);
>> [~, PO_MAX, ind] = ode_isol2rpp(prob, 'po', @tube, @tube_DFDX, @tube_DFDP,...
     t0, x0, p, p0, '−zpp', 'zero_phase');
```

First a cell-array of strings `p` is assigned to the system parameters of `tube`, then the continuation problem structure is initialized in the call to `coco_prob`. Next, we make a call to the function `orbit settings` to define some standardized settings for continuation problems of periodic orbits; see subsection SM1.4.1. Finally, the call to the `ode_isol2rpp` constructor constructs the continuation problem; as is uncommon for CoCo constructor functions it also initiates the continuation and returns the index of the maximum in the first coordinate of the periodic orbit. The constructor takes as arguments the continuation problem structure `prob`, function handles for the vector field and (optionally) its Jacobians with respect to state variables and

parameters, the elements `t0` and `x0` of the trajectory segement defining Γ^s, the cell array of names of system parameters `p` and the values `p0`. While locating the maximum in the first coordinate of Γ^s, the end point of the phase segment $\boldsymbol{x_1}$ is fixed, its start point $\boldsymbol{x_1}(0)$ is allowed to move around Γ^s subject to periodicity conditions with the anchoring segment $\boldsymbol{x_2}(t)$. CoCo's event location functionality is used to determine solutions to the continuation problem when $\boldsymbol{x_1}(0)$ moves through a local extrema of Γ^s by detecting when `tube(`$\boldsymbol{x_1}(0)$`, p0)`$\cdot\binom{1}{0} = 0$. These solutions are given the type `XTRM` and may be searched for with the function `coco_bd_labs`. In the output of this call to `ode_isol2rpp` below, we see columns for the phase θ relative to γ_0, the time derivative of the first coordinate of the system, and state variables at the start point $\boldsymbol{x_1}(0)$, labeled as `theta`, `phi0`, `x0_1` and `x0_2`, respectively.

STEP	TIME	‖U‖	LABEL	TYPE	theta	phi0	x0_1	x0_2
0	00:00:01	1.7343e+01	1	EP	−1.9440e−15	4.1529e−01	1.9247e−01	−5.7804e−01
21	00:00:03	1.5392e+01	2	FP	1.5143e−02	4.2857e−01	1.2538e−01	−5.2991e−01
273	00:00:22	1.2174e+01	3	XTRM	3.6063e−01	2.7756e−17	−4.7211e−01	4.0244e−01
397	00:00:31	1.1802e+01	4	FP	5.1514e−01	−4.2857e−01	−1.2538e−01	5.2991e−01
643	00:00:48	1.2753e+01	5	XTRM	8.6063e−01	1.3878e−17	4.7211e−01	−4.0244e−01
729	00:00:54	1.3949e+01	6	EP	1.0000e+00	4.1529e−01	1.9247e−01	−5.7804e−01

It follows that the solution which has $\boldsymbol{x_1}(0)$ at the maximum of `x0` is the one, marked `XTRM`, with the largest value in the `x0_1` column. The label number for this solution is returned in the variable `ind`, as determined by the call to `ode_isol2rpp` above. This solution is read from the output files via `rpp_read_solution`, as below, using the option `'zero phase'` to insure that the function concatenates $\boldsymbol{x_1}(t)$ and $\boldsymbol{x_2}(t)$ in the correct order.

```
>> sol = rpp_read_solution('po', 'zero_phase', ind);
>> t0 = sol.tbp;
>> x0 = sol.xbp;
```

The returned structure `sol` contains a number of fields, including the array of time instances `tbp` corresponding to the array of state variable values `xbp`, the array of system parameter values `p`, and the phase `theta` relative to the start point of the original periodic orbit.

SM1.4.4. FLQT: Computing the Floquet bundle. The `'flqt'` toolbox is designed to facilitate the computation of Floquet multipliers and Floquet bundles of periodic orbits for use as linear approximations of isochrons. Floquet multipliers of a periodic orbit Γ, as well as a numerical approximation of its Floquet bundle, are computed from the variational equations. The methods for computing Floquet multipliers and bundles in [SM2] are encoded in the constructor `ode_isol2flqt` and the utility `flqt_read_solution` for ease of access. Further, the constructor uses the `'rpp'` toolbox to facilitate the rotation of the Floquet bundle along with a periodic orbit, such that the Floquet vector for any point $\gamma_\theta \in \Gamma$ may be determined accurately without first rotating the periodic orbit and recomputing the Floquet bundle, or the need for interpolation along the Floquet bundle.

Computing a Floquet vector. From the solutions stored in `t0` and `x0` obtained in subsection SM1.4.3 we compute the Floquet multipliers μ and Floquet bundle $\boldsymbol{w}(t)$ of Γ^s such that $\boldsymbol{w_0} = \boldsymbol{w}(0)$ is the zero-phase Floquet vector at γ_0. The following commands encode the continuation problem; note that the function `coco` is called to perform continuation once the continuation problem structure `prob` is constructed. In the call to `coco` a `0`-dimensional mani-

fold is requested; rather than performing continuation, a single solution to the boundary value problem encoded by `ode_isol2flqt` is found by CoCo's Newton solver by starting from the supplied initial condition (`t0`, `x0`, and `p0`).

```
>> prob  = coco_prob();
>> prob  = orbit_settings(prob);
>> prob  = ode_isol2flqt(prob, 'po', @tube, @tube_DFDX, @tube_DFDP,...
     t0, x0, p, p0);
>> FLQT0 = coco(prob, 'floquet0', [], 0);
>> sol0  = flqt_read_solution( 'po', 'floquet0', 1 );
```

The solution is read using the utility `flqt_read_solution`, producing a structure `sol0` with fields including the array of time instances `tbp` corresponding to the array of state variable values `xbp`, the array of system parameter values `p`, the Floquet multiplier `mu`, and the Floquet vector `w` associated with γ_0. With this solution, one may begin computation of the zero-phase isochron $I_0(\Gamma^s)$ of Γ^s. However, should the reader wish to compute a foliation of isochrons, `ode_isol2flqt` is able to rotate the Floquet vector computed at γ_0, along with Γ^s to obtain a set of distinct phases.

Rotating the Floquet bundle. The `'flqt'` toolbox extends the `'rpp'` toolbox to encode a larger continuation problem structure allowing for the rotation of both a periodic orbit and its Floquet bundle. The periodic orbit, and its Floquet bundle are separated into the phase and anchoring segments, per subsection SM1.1; the Floquet bundle is subject to constraints as presented in [SM6] at $\boldsymbol{x_1}(\theta T_\Gamma) = \gamma_\theta$, and periodicity conditions at $\boldsymbol{x_1}(0) = \gamma_0$. With the solution `sol0` from above, the following commands may be used to encode a continuation problem to rotate the Floquet vector around Γ^s.

```
>> theta = 0 : (1 / 20) : 0.99;
>> prob  = coco_prob();
>> prob  = orbit_settings(prob);
>> prob  = ode_isol2flqt(prob, 'po', @tube, @tube_DFDX, @tube_DFDP,...
     sol0.tbp, sol0.xbp, p, sol0.p, theta, '-flqt0', {sol0.mu; sol0.w});
>> FLQT  = coco(prob, 'floquet', [], 1, {'theta', 'mu', 'wth_1', 'wth_2'},...
         {[-1e-4, max(theta) + 1e-4], [], [], []});
```

First, an array of phases for which Floquet vectors are desired is set in the variable `theta`; note that due to periodicity the phase $\theta = 1$ is not included. The following two lines initialize the continuation problem structure and its settings, and the call to `ode_isol2flqt` encodes the Floquet vector rotation problem. After supplying function handles for (10) and its Jacobians, the time and state variable arrays `sol0.tbp` and `sol0.xbp` defining Γ^s, the array strings `p` and values `sol0.p` for systems parameters, we supply the array of phases `theta`, and the stable Floquet multiplier and its Floquet vector in the array `{sol.mu; sol.w}` following the option `'-flqt0'`. Continuation is initiated by the call to `coco`; here, `1` is input to request a one-dimensional set of solutions to the continuation problem, the array of strings `{'theta', 'mu', 'wth_1', 'wth_2'}` specify the variables allowed to change during continuation, and `{[-1e-4, max(theta)+ 1e-4], [], [], []}` sets the computational bounds for each continuation variable represented by a string in the previous array. The principle continuation parameter is chosen as `'theta'`, the phase associated with points $\gamma_\theta \in \Gamma$. The subsequent free

continuation parameters 'mu', 'wth_1', and 'wth_2', represent the Floquet multiplier and the coordinates of the Floquet vector μ and $\boldsymbol{w}_{\boldsymbol{\theta}}$, respectively. These strings, or replacement strings representing these variables (type `help ode_isol2flqt` into MATLAB for details), are required when rotating a Floquet bundle. The computational bounds on 'theta' should be set slightly outside of $[0, \max(\texttt{theta})]$ to avoid the initial solution given by `t0` and `x0` converging outside of computational bounds. Each solution corresponding to a phase in `theta` is located by CoCo's event location algorithm and marked with the type `THETA`, as seen in the screen output below.

STEP	TIME	‖U‖	LABEL	TYPE	theta	mu	wth_1	wth_2
0	00:00:00	1.7723e+01	1	EP	0.0000e+00	1.3446e−03	−2.8737e−01	9.5782e−01
1	00:00:00	1.7723e+01	2	THETA	0.0000e+00	1.3446e−03	−2.8737e−01	9.5782e−01
1	00:00:00	1.7724e+01	3	EP	−1.0000e−04	1.3446e−03	−2.8667e−01	9.5803e−01

STEP	TIME	‖U‖	LABEL	TYPE	theta	mu	wth_1	wth_2
0	00:00:00	1.7723e+01	4	EP	0.0000e+00	1.3446e−03	−2.8737e−01	9.5782e−01
46	00:00:11	1.4566e+01	5	THETA	5.0000e−02	1.3446e−03	−5.7738e−01	8.1647e−01
88	00:00:20	1.4302e+01	6	THETA	1.0000e−01	1.3446e−03	−7.5328e−01	6.5770e−01
132	00:00:28	1.3984e+01	7	THETA	1.5000e−01	1.3446e−03	−8.5442e−01	5.1958e−01
181	00:00:37	1.3695e+01	8	THETA	2.0000e−01	1.3446e−03	−9.1382e−01	4.0611e−01
235	00:00:47	1.3536e+01	9	THETA	2.5000e−01	1.3446e−03	−9.5274e−01	3.0380e−01
297	00:00:57	1.3280e+01	10	THETA	3.0000e−01	1.3446e−03	−9.9504e−01	9.9447e−02
364	00:01:08	1.3028e+01	11	THETA	3.5000e−01	1.3446e−03	−8.7623e−01	−4.8189e−01
417	00:01:18	1.2518e+01	12	THETA	4.0000e−01	1.3446e−03	−4.9938e−01	−8.6638e−01
464	00:01:26	1.2406e+01	13	THETA	4.5000e−01	1.3446e−03	−1.0337e−01	−9.9464e−01
508	00:01:32	1.2485e+01	14	THETA	5.0000e−01	1.3446e−03	2.8737e−01	−9.5782e−01
548	00:01:38	1.2469e+01	15	THETA	5.5000e−01	1.3446e−03	5.7738e−01	−8.1647e−01
585	00:01:44	1.2580e+01	16	THETA	6.0000e−01	1.3446e−03	7.5328e−01	−6.5770e−01
622	00:01:52	1.2822e+01	17	THETA	6.5000e−01	1.3446e−03	8.5442e−01	−5.1958e−01
661	00:01:58	1.2926e+01	18	THETA	7.0000e−01	1.3446e−03	9.1382e−01	−4.0611e−01
706	00:02:05	1.3264e+01	19	THETA	7.5000e−01	1.3446e−03	9.5274e−01	−3.0380e−01
759	00:02:13	1.3325e+01	20	THETA	8.0000e−01	1.3446e−03	9.9504e−01	−9.9447e−02
821	00:02:22	1.3433e+01	21	THETA	8.5000e−01	1.3446e−03	8.7623e−01	4.8189e−01
871	00:02:29	1.3655e+01	22	THETA	9.0000e−01	1.3446e−03	4.9938e−01	8.6638e−01
923	00:02:38	1.4357e+01	23	THETA	9.5000e−01	1.3446e−03	1.0337e−01	9.9464e−01
923	00:02:38	1.4359e+01	24	EP	9.5010e−01	1.3446e−03	1.0255e−01	9.9473e−01

The rotated Floquet vector solutions are read with the utility `flqt_read_solution`. Each solution may be stored individually, though when intending to compute a foliation of isochrons it is best to store them in an array, which is achieved by the following.

```
>> THTlabs = coco_bd_labs(FLQT, 'THETA');
>> Nphase  = numel(THTlabs);
>> sol     = cell(Nphase, 1);
>> for phi = 1 : Nphase
     sol{phi} = flqt_read_solution('po', 'floquet', THTlabs(phi));
>> end
```

Here, the first line produces an array of label numbers for each solution with the type `THETA`, which is measured to determine the number solutions to be read and isochrons to be computed. Each solution `sol{phi}` is a structure with fields including the array of time instances `tbp` corresponding to the arrays of state variable values `xbp` and Floquet bundle `wbp`, the array

of system parameter values `p`, the phase `theta` of the solution relative to γ_0, the Floquet multiplier `mu`, and the Floquet vector `w` associated with γ_θ.

SM1.4.5. ISCRN: Computing an isochron of Γ^s. The `'iscrn'` toolbox is designed to faciliate the computation of isochrons of planar systems as smooth one-dimensional manifolds by the application of the boundary value problem set up described in subsection SM1.1. As a guide for the use of the toolbox, we now explain the use of the constructors of the `'iscrn '`. While the guide in this section may simply be repeated to compute an isochron foliation, the script `isochron_foliation.m` is the folder `iscrn/help` provides an example of how to implement the computation of many isochrons of the respective foliation.

Computing the fundamental domain. We begin with the computation of the fundamental arc $I_0^0(\Gamma^s)$ of $I_0(\Gamma^s)$ in region $\boldsymbol{D}$, and hence the computation of the fundamental domain $\boldsymbol{s_0}$ of $I_0(\Gamma^s)$. We use the solution structure `sol0` returned from `flqt_read_solution` in subsection SM1.4.4 to set up the continuation problem below, starting by setting $\delta_{\max}$, the maximal distance from the periodic orbit, that bounds `deltamax`. The choice of this value governs the accuracy of the approximation of $I_0(\Gamma^s)$; while it should be chosen sufficiently small, we note that excessively small values of $\delta_{\max}$ lead to computational issues.

```
>> deltamax = 5e-4;
>> prob = coco_prob();
>> prob = iscrn_settings(prob);
>> prob = coco_set(prob, 'cont', 'ItMX', [0, 100]);
```

The function `iscrn_settings` is supplied in subsection SM1.4.1, and we note here that the choice of `'NTST'` should be made carefully considering the guidance given there. After invoking `iscrn_settings`, the following line instructs CoCo to perform continuation only in the forward direction with respect to the primary continuation parameter. This setting prevents CoCo from moving the end point $\boldsymbol{x}(T_{\Gamma^s})$ backwards along $\boldsymbol{w_0}$, per default settings.

The fundamental arc of $I_\theta(\Gamma^s)$ is computed via the following commands.

```
>> prob = ode_isol2iscrn(prob, 'T0', @tube, @tube_DFDX, @tube_DFDP,...
     sol0.tbp, sol0.xbp, p, sol0.p, sol0.w, [],...
     '-fund', {deltamax; sol0.mu}, '-arc', {0, 20});
>> ISO{1} = coco(prob, 'Gs/arc00+', [], 1,...
     {'tau', 'delta', 'x0_1', 'x0_2', 'l'},...
     {[], [-1, 1], [], [], []});
```

The string `'T0'` is the identifier allocated to the trajectory segement $\boldsymbol{x}(t)$, and will be used later to read data from saved files. The solution structure `sol0` from subsection SM1.4.4 contains a numerical approximation of Γ^s, which is used as the initial solution for the BVP, as well as a copy of the system parameters `sol0.p`. Additionally, `sol0` contains the fields `mu` and `w` for the Floquet multiplier μ and vector $\boldsymbol{w_0}$, respectively. These are passed to `ode_isol2iscrn` as above; to compute the other branch of $I_\theta(\Gamma^s)$ (on the other side of Γ^s) `sol.w` is multiplied by -1. The empty array `[]` following this is a dummy variable that is ignored when the option `'-fund'` is invoked to compute the fundamental domain. This option is followed by the cell array `{deltamax; sol0.mu}` that supplies our choice of $\delta_{\max}$ and the stable Floquet multiplier of Γ^s. The option `'-arc'` instructs CoCo to measure the arclength along the numerical

approximation of $I_0(\Gamma^s)$ during continuation; it must be followed by a cell array that contains the bounds, in terms of arclength, at which the computation should stop when met. While we do not expect a large arclength to be covered by the fundamental arc of $I_0(\Gamma^s)$, this option should be used nonetheless if one intends to measure the arclength along subsequent arcs of $I_0(\Gamma^s)$.

The first argument of the call to `coco`, the string `'Gs/arc00+'`, names the directory in which output files are stored. We advise using the format

```
'<isochron name><isochron number>/<arc name><arc number><side>',
```

where `'side'` is either the string `'−'` or `'+'`, to construct this name such that the output files for each arc are sorted into directories of their respective isochron, and also not overwritten. The post-processing functions provided in subsection SM1.6 to read these files assume this format. The cell array of continuation parameters `{'tau', 'delta', 'x0_1', 'x0_2', 'l'}` must be included, although the final string may be omitted if the option `'−arc'` is not being used. The following cell array of bounds for the continuation parameters `{[], [−1, 1], [], [], []}` is also required, even though we are only concerned with setting the computational bounds of `delta` when computing the fundamental arc of $I_\theta(\Gamma^s)$. The constructor `ode_isol2iscrn` rescales both continuation variables `tau` and `delta` by $\sqrt{\mu\,\delta_{\max}}$ and $\delta_{\max}$, respectively. As a result, we set the computational bounds on `delta` to ± 1 instead of $\pm\delta_{\max}$. The rescaling of `tau` attempts to improve computational stability by predicting the value of `tau` at which `delta` $= \delta_{\max}$. True values for `tau` and `delta` can be computed by multiplying each by $\sqrt{\mu\,\delta_{\max}}$ and $\delta_{\max}$, respectively.

We advise to check CoCo's screen output, making sure that the continuation has terminated with both a positive value of `tau` and `delta` $= \pm 1$, as below.

STEP	TIME	||U||	LABEL	TYPE	tau	delta	x0_1	x0_2	l
0	00:00:00	1.3782e+01	1	EP	0.0000e+00	0.0000e+00	4.7211e−01	−4.0244e−01	0.0000e+00
38	00:00:02	1.4112e+01	2	EP	9.7594e−01	−1.0000e+00	4.6565e−01	−3.8264e−01	2.0825e−02

Should the terminal value of `tau` be negative, then the computed fundamental domain $\boldsymbol{s_0}$ belongs to the other branch of $I_0(\Gamma^s)$. The preceding commands should then be repeated, except that `prob = coco_set(prob, 'cont', 'ItMX', [100, 0])` should replace the appropriate line in order to initiate backward continuation with respect to `tau`. If instead the terminal value for `delta` has absolute value less than one, the above commands should also be repeated with a higher value given for the non-zero number following `'ItMX'`.

In addition to being saved to file, the above output is assigned to the variable `ISO{1}`, and any continuation parameter may be extracted from `ISO{1}` using standard the CoCo function `coco_bd_col` as below, where the final argument passed to this function is a string or cell array of strings of names of the continuation parameters that one wishes to extract; the names of these columns can be quickly queried via `ISO{1}{1,:}`, but this is beyond what this guide aims to explain.

```
>> tau   = coco_bd_col(ISO{1}, 'tau');
>> delta = coco_bd_col(ISO{1}, 'delta');
>> x0    = coco_bd_col(ISO{1}, {'x0_1', 'x0_2'});
```

The change in the scaled orthogonal deviation `delta` from the linear approximation of $I_\theta(\Gamma^s)$, while the scaled distance along that linear approximation `tau` is varied, is viewed with the following commands.

```
>> plot(tau, delta)
>> xlabel('tau')
>> ylabel('delta')
```

The short arc of $I_\theta(\Gamma^s)$ computed while locating the fundamental domain is thus viewed by the following commands.

```
>> plot(x0(1,:), x0(2,:))
>> xlabel('x')
>> ylabel('y')
```

Alternatively, one may use the provided reader `iscrn_read_arc` instead of `coco_bd_col` to extract this data; this function is covered in subsection SM1.6.

Computing subsequent arcs of the isochron. Computing the further arcs of $I_0(\Gamma^s)$ is best achieved via the constructor `ode_iscrn2iscrn`. This constructor passes the tangent vector to the continuation problem to ensure smoothness along the boundaries of adjacent isochron arcs. In this section we show the computation of the first and second arcs $I_0^1(\Gamma^s)$ and $I_0^2(\Gamma^s)$ of the isochron, using the results of the computation of the fundamental arc $I_0^0(\Gamma^s)$. These instructions may be altered to compute further arcs of $I_0(\Gamma^s)$.

We first choose a reasonable number of mesh points `ntst` for the trajectory segement $\boldsymbol{x}(t)$ that implement the map defining the isochron. As `iscrn_settings` turns the `'NAdapt'` option on the reader is advised to carefully set the `'NTST'` and `'NTSTMX'` settings as below. Specifically, `'NTST'` should be set to match the actual number of mesh points used in the solution to start the continuation, and `'NTSTMX'` should be set as the desired maximum number of mesh points for $\boldsymbol{x}(t)$ as $k \cdot$ `ntst`. Below we choose `ntst` as 50, which is relatively high for the observed geometric complexity of this isochron. The second line extracts the true number of mesh points used in the final solution of the previous continuation run, stored in the `'T0.iscrn.NTST'` column of `ISO{1}` at label number `2`, using the post-processor function `coco_bd_val`. This string is constructed as `'<object identifier>.<toolbox>.NTST'`, referring to the object identifier that we set in the call to `ode_isol2iscrn` in the paragraph on the fundametal domain in subsection SM1.4.5.

```
>> ntst   = 50;
>> s_NTST = coco_bd_val(ISO{1}, 2, 'T0.iscrn.NTST');
>> prob   = coco_prob();
>> prob   = iscrn_settings(prob);
>> prob   = coco_set(prob, 'coll', 'NTST', s_NTST, 'NTSTMX', ntst);
>> prob   = ode_iscrn2iscrn(prob, 'T1', 'Gs/arc00+', 'T0', 2, 'Gs/arc00+', 'T0', 2);
>> ISO{2} = coco(prob, 'Gs/arc01+', [], 1, {'tau', 'x0_1', 'x0_2', 'l'},...
     {[-0.1, 1], [-1,1], [-1,1], []});
```

The next two lines initialize the continuation problem and its settings, and the following line sets the number of mesh points as described above. Specifically, for the first arc of $I_0(\Gamma^s)$ we move the end point $\boldsymbol{x}(T_{\Gamma^s})$ over the fundamental domain $\boldsymbol{s_0}$. Since we use the time-T_{Γ^s} map to compute $I_0(\Gamma^s)$ we do increase the number of mesh points of the continuation problem.

The call to `ode_iscrn2iscrn` takes the continuation problem `prob`, followed by the object identifier `T1` that identifies this construction. The following three inputs refer to the solution from which to start the computation of the isochron, in this case the terminal solution of the fundamental arc. Specifically, the directory storing the solution `'Gs/arc00+'`, the object identifier for the solution `'T0'`, and the label number of that solution `2`. The final three input arguments refer again to the terminal solution of the fundamental arc, but now to identify the solution $\boldsymbol{s}(t)$ that defines the fundamental domain $\boldsymbol{s_0}$. While these arguments are indeed not necessary to construct the continuation problem for the first arc, all subsequent arcs of $I_0(\Gamma^s)$ append $\boldsymbol{s}(t)$ to the end of the terminal solution of the previous isochron arc in order to construct their starting solution. The function `coco` initiates the continuation, and stores the output in `ISO{2}`, saving files to `'Gs/arc01+'`. The arguments to this function are as is subsection SM1.4.5, except for the exclusion of `delta` and its corresponding bounds, as well as the addition of computational boundaries to `tau`. Again `tau` is scaled such that its continuation should terminate when `tau` $= 1$ and $\boldsymbol{x}(T_{\Gamma^s})$ reaches the end of $\boldsymbol{s_0}$, while the left boundary is set past 0 to avoid the initial solution not converging. Computational bounds on the arclength `l` of $I_0(\Gamma^s)$ are read from the previous solution, and automatically passed to the constructor.

Constructing the second arc $I_0^2(\Gamma^s)$ of the isochron is achieved via commands very similar to those above, except that the number of mesh points should be increased. The first line below shows the command required to extract the number of mesh points in the terminal solution stored in `ISO{2}`. Note that we reference the object identifier `'T1'` as set above to name the trajectory segment that solves the BVP defining $I_0^1(\Gamma^s)$. The continuation problem is then initialized, and has its settings set, followed by the careful setting of the number of mesh points to be used in computing the second arc of $I_0(\Gamma^s)$. We set the true number of mesh points `'NTST'` as `k_NTST + s_NTST`, the number of mesh points of the terminal solution in the previous arc added to the number of mesh points in the terminal solution of the fundamenal arc, respectively. Given that we now use the time-$2T_{\Gamma^s}$ map to compute $I_0(\Gamma^s)$, we increase the maximum number of mesh points to $2 \cdot$ `ntst`.

```
>> k_NTST = coco_bd_val(ISO{2}, 2, 'T1.iscrn.NTST');
>> prob   = coco_prob();
>> prob   = iscrn_settings(prob);
>> prob   = coco_set(prob, 'coll', 'NTST', k_NTST + s_NTST, 'NTSTMX', 2 * ntst);
>> prob   = ode_iscrn2iscrn(prob, 'T2', 'Gs/arc01+', 'T1', 2,...
     'Gs/arc00+', 'T0', 2);
>> ISO{3} = coco(prob, 'Gs/arc02+', [], 1, {'tau', 'x0_1', 'x0_2', 'l'},...
     {[-0.1, 1], [-1,1], [-1,1], []});
```

In the call to `ode_iscrn2iscrn` we pass the continuation problem structure `prob`, and supply the object identifier `'T2'` of the trajectory segment that represents the new map for $I_0^2(\Gamma^s)$. The following three arguments refer to the terminal solution of the previous arc, which will be concatenated with the terminal solution of the fundamental arc, as identified by the next three arguments, to construct the initial solution for the continuation problem to compute $I_0^2(\Gamma^s)$. The following call to `coco` initiates continuation, storing output in `ISO{3}` and in files in the directory `'Gs/arc00+'`. Its other arguments remain as explained above.

Subsequent arcs of $I_0(\Gamma^s)$ may be computed by repeating the commands above, but changing arguments so that the terminal solution of the second arc of the isochron is used and the

data is stored in the appropriate directory. Care should be taken to update the number of mesh points as specified. The other branch of$I_0(\Gamma^s)$ is likewise computed by following the instructions in this section after changing the sign of the Floquet vector when computing the fundamental domain, as discussed in subsection SM1.4.5. A foliation of isochrons is computed by following these instruction for each solution returned by `ode_isol2flqt` as described in subsection SM1.4.4. Alternatively, the script `isochron_foliaiton.m` located in the folder `iscrn/examples` may be used to automate that process.

To visualize the results of these computations, follow the directions at the end of the paragraph on the fundametal domain in subsection SM1.4.5, or see subsection SM1.6.

SM1.5. Computing isochrons of a repelling focus equilibrium. Isochrons may be defined and computed for focus-type equilibria such as the repellor q_+ in region $\boldsymbol{D}$; the required code discussed in this sections is in the file `ep_backward_demo.m` in `iscrn/toolbox/examples/tube`. The approach of continuing the solution to a BVP defining the time-T_{q+} map is identical to that for periodic orbits and the only difference lies in computating the linear approximation of such isochrons. We now guide the reader through the computation of the first three arcs of the zero-phase backward-time $U_0(q_+)$, which, notably, also is an example of using the `'-u'` option of the `'iscrn'` toolboxes when computing backward-time isochrons. Given the similarity in procedure and steps with those in subsection SM1.4, we keep this section quite brief.

SM1.5.1. Computing the linear approximation of an isochron of the focus. The function `ep_bundle` is used when computing a set of linear approximations of isochrons of a focus equilirbium q for phases θ stored in an array `theta`. This function constructs the ellipse that represents a blow-up about q [SM8], and returns the linear approximation in a structure that mimics that returned by `flqt_read_solution`. The commands below return a cell-array of structures `sol` given an array of phases `theta`.

```
>> theta   = 0 : (1 / 20) : 0.99;
>> Nphase  = numel(THTlabs);
>> sol     = cell(Nphase, 1);
>> for phi = 1 : Nphase
>>   sol{phi} = ep_bundle(@tube_DFDX, x0, p0, 5e-4, theta(phi));
>> end
```

Each call to `ep_bundle` requires the handle to the Jacobian with respect to the state variables of the system, the equilibrium `x0`, the array of parameter values `p0`, a scaling variable for the magnitude of the ellipse, and a phase `theta(phi)` for which the linear approximation of the isochron is requested.

SM1.5.2. Computing an isochron. The initial solution requires a trajectory segement, in this case the steady-state solution, and the linear approximation of the isochron as above. The real part of the equilibrium's eigenvalues are used in place of the Floquet multiplier for determining a scale on the linear approximation of the isochron, while T_{q+} is given by the complex part of the focus' eigenvalues. While the subsequent computations follow those in subsection SM1.4.5, we go over them here briefly as well to demonstrate the use of the `'-u'` option used when computing backwards-time isochrons.

Computing the fundamental domain. For the computation of the fundamental arc $U_0^0(q_+)$ of $U_0(q_+)$ in region $\boldsymbol{D}$, and hence the computation of the fundamental domain $\boldsymbol{s_0}$ of $U_0(q_+)$, we set `deltamax` as in subsection SM1.4.5, and invoke `iscrn_settings`. The next line instructs CoCo to perform continuation only in the forward direction with respect to the primary continuation parameter.

```
>> deltamax = 5e-4;
>> prob = coco_prob();
>> prob = iscrn_settings(prob);
>> prob = coco_set(prob, 'cont', 'ItMX', [0, 100]);
```

The fundamental arc of $U_0(q_+)$ is computed via the following commands, where the arguments for `ode_isol2iscrn` are as described in subsection SM1.4.5 and the final option `'-u'` instructs the constructor to reverse time of the vector field and its Jacobians; in other words, a backward-time isochron is computed as a forward-time isochron of the time-reversed system.

```
>> prob = ode_isol2iscrn(prob, 'T0', @tube, @tube_DFDX, @tube_DFDP,...
     sol.tbp, sol.xbp, p, sol.p, sol.w, [],...
     '-fund', {deltamax; sol.mu}, '-arc', {0, 20}, '-u');
>> ISO{1} = coco(prob, 'ep/arc00+', [], 1,...
     {'tau', 'delta', 'x0_1', 'x0_2', 'l'},...
     {[], [-1, 1], [], [], []});
```

Likewise, the arguments given to `coco` are as in subsection SM1.4.5, except that output files are instead instructed to be saved to `'ep/arc00+'`. Also, here the value of `tau` is rescaled as $\sqrt{\alpha\,\delta_{\max}}$. Again, we advise the user to check CoCo's screen output to make sure that the continuation has terminated with positive values of `tau` and `delta` $= \pm 1$, as below.

STEP	TIME	||U||	LABEL	TYPE	tau	delta	x0_1	x0_2	l
0	00:00:00	1.1565e+01	1	EP	0.0000e+00	2.2710e-02	3.2526e-01	-2.4137e-01	5.2456e-05
55	00:00:04	1.0413e+01	2	EP	1.5742e+00	1.0000e+00	3.4932e-01	-2.6616e-01	3.4608e-02

Should the terminal value of `tau` be negative, then the computed fundamental domain $\boldsymbol{s_0}$ is that of $U_{0.5}(q_+)$. The preceding commands should then be repeated, except that `prob = coco_set(prob, 'cont', 'ItMX', [100, 0])` should replace the appropriate line in order to initiate backward continuation with respect to `tau`. If instead the terminal value for `delta` has absolute value less than one, the above commands should also be repeated with a higher value for the non-zero number following `'ItMX'`.

Computing further arcs of $U_0(q_+)$. Computing the further arcs of $U_0(q_+)$ is achieved via the constructor `ode_iscrn2iscrn`, which passes the tangent vector to the continuation problem to ensure smoothness across adjacent isochron arcs. This constructor also passes the option `'-u'`, and hence this option is not required for further arcs of $U_0(q_+)$. Instructions for the computation of a the foliation $\mathcal{U}(q_+)$ of backward-time isochrons of q_+ are identical to those for the computation of $\mathcal{I}(\Gamma^s)$ given in subsection SM1.4.5. Note from the comments in `iscrn_settings.m` that the absolute tolerance and number of mesh points required to compute $U_{0.4}(q_+)$ are reduced and increased, respectively, to ensure that CoCo's Newton solver converges; see also subsection SM1.4.1. A visualization of a computed foliation can be found in subsection SM1.6.3.

SM1.6. Extracting isochron data. We provide post-processing functions to read solutions from an isochron continuation, as well as a set of functions that process the isochron from the files stored in the `data` directory to which CoCo saves output files, or a different directory in the MATLAB path. Each of these functions assumes that such files are named in the format:

```
'<isochron name><isochron number>/<arc name><arc number><side>'
```

This is indeed the case when following the instructions in this guide. Eeven when moving isochron files out of the `data` directory, as long as they have the above format and are in the MATLAB path, the post-processing functions discussed in this section will extract the data as desired.

SM1.6.1. Extracting trajectory segements. The trajectory segement that solves the time-kT map associated with the isochron computation for a given label is read from file by the post-processor `iscrn_read_solution`. The following commands read the terminal solution of the fundamental arc on the isochron `'Gs/arc00+'`, computed per instructions in subsection SM1.4.5 and hence identified by `'T0'` with terminal solution label 2.

```
>> sol = iscrn_read_solution('T0', 'Gs/arc00+', 2);
```

The resulting solution structure `sol` has a number of useful fields, including the representation of $\boldsymbol{x}(t)$ in the fields `tbp` and `xbp`, the state parameter values in the field `p`, the normalized fundamental domain and its original length in the fields `s` and `tau_max`, and the true number of mesh points used for this solution in `NTST`. Other fields are included in this structure, and they are explained in the help section of `iscrn_read_solution`.

SM1.6.2. Extracting isochrons. The post-processing function `iscrn_read_isochron` extracts the one-dimensional curve representing the isochron as the union of the computed isochron arcs. In the example below we extract one branch of the zero-phase isochron of $\mathcal{I}(\Gamma^s)$ as computed following the instructions in subsection SM1.4.5. The isochron name format `'Gs/arc%02d+'` is passed as the first argument, followed by the object identifier format `'T\%d'`. These strings are formatted as described in [SM11]. The final argument is a cell array of names of continuation variables that the reader wishes to extract. In this case, the coodinates of the isochron. The option `'-trunc'` indicates that either the `'initial'` or `'final'` points of each arc of the isochron should be truncated during concatenation, since the first and last points on successive arcs of an isochrons are near identical.

```
>> iscrn = iscrn_read_isochron('Gs/arc%02d+', 'T%d', {'x0_1', 'x0_2'},...
     '-trunc', 'initial');
```

The structure `iscrn` has a number of fields, of which `iscrn` is of primary interest. The requested variables in the cell array can be accessed in `iscrn.iscrn`, which is returned as a matrix with columns assigned to each variable requested. If the option `'-arc'` is used during computation, the field `l` contains the computed arclength for each point along the isochron. Additional fields include the periodic orbit via `tpo` and `xpo`, the trajectory segments defining the fundamental domain `tbp_s` and `xbp_s`, state parameters in `p`, the stable Floquet multiplier and its Floquet vector in `mu` and `w`, respectively, and the normalized fundamental domain and

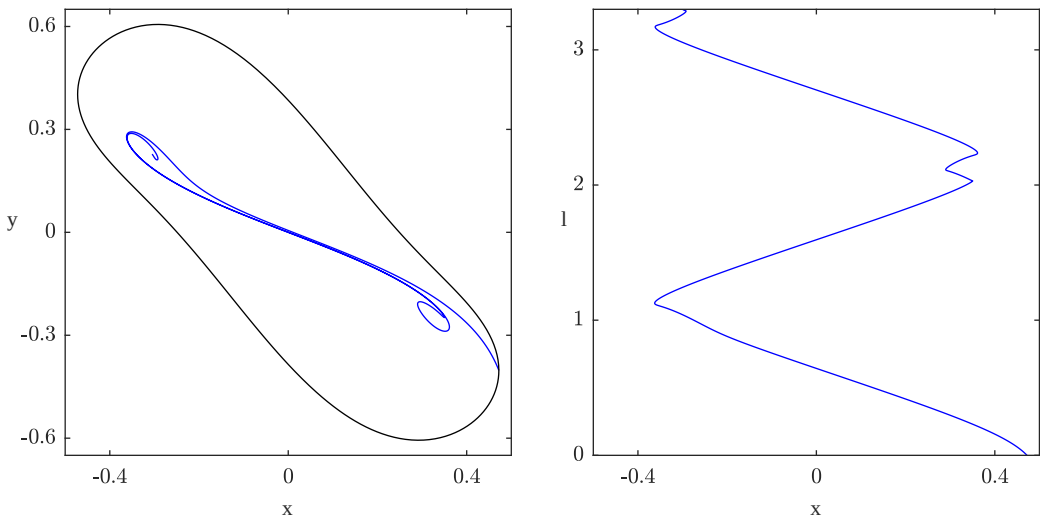

Figure SM1. *Left: The branch of the zero-phase isochron $I_0(\Gamma^s)$ (blue) in region $\boldsymbol{D}$ inside of Γ^s (black) as computed in subsection SM1.4.5. Right: Arclength along $I_0(\Gamma^s)$ in region $\boldsymbol{D}$ against its x-coordinate; compare with subsection 3.5.*

its original norm in `s` and `taumax`, respectively, along with various other fields as documented in the help section of `iscrn_read_isochron`.

Visualisation of this branch of $I_0(\Gamma^s)$, along with Γ^s is achieved by the following commands, together with a separate plot of the arclength of $I_0(\Gamma^s)$ against its x-coordinate.

```
>> figure(1)
>> plot(iscrn.iscrn(:,1), iscrn.iscrn(:,2), 'b')
>> hold on
>> plot(iscrn.xpo(:,1), iscrn.xpo(:,2), 'k')
>> xlabel('x')
>> ylabel('y', 'rotation', 0)
>> figure(2)
>> plot(iscrn.iscrn(:,1), iscrn.l, 'b')
>> xlabel('x')
>> ylabel('l', 'rotation', 0)
```

The resulting plots are shown in Figure SM1.

Extracting arcs of isochrons. Particularly when troubleshooting an isochron computation, it may be useful to extract only the data of a single arc with `iscrn_read_arc`. The example below extracts the fundamental arc of the zero-phase isochron of $\mathcal{I}(\Gamma^s)$ as computed by following the instructions in subsection SM1.4.5. The isochron name `'Gs/arc00+'` is passed as the first argument, followed by the object identifier `'T0'`, then a cell array of names of continuation variables that the reader wishes to extract, in this case, the coodinates of the isochron.

```
>> arc = iscrn_read_arc('Gs/arc00+', 'T0', {'x0_1', 'x0_2'});
```

The structure `arc` contains similar fields as those of `iscrn` from subsection SM1.6.2; information on further fields is contained in the help section of `iscrn_read_arc`.

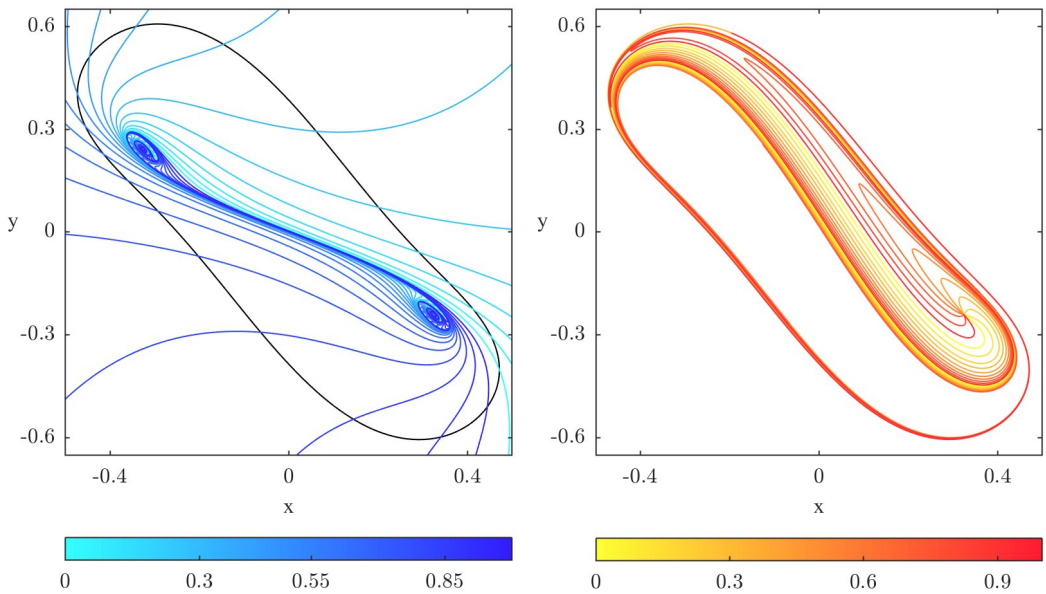

Figure SM2. *Two foliations of isochrons computed using the supplied script* `isochron_foliation.m`. *Left: Twenty forward-time isochrons (cyan–blue) of the foliation* $\mathcal{I}(\Gamma^s)$ *of* Γ^s *(black) in region* $\boldsymbol{D}$*, equally spaced in phase and colored according to the color bar. Right: Ten backward-time isochrons (yellow–red) of the foliation* $\mathcal{U}(q_+)$ *of* q_+ *in region* $\boldsymbol{D}$*, equally spaced in phase and colored according to the color bar. Compare with subsection 3.5.*

SM1.6.3. Extracting an entire isochron foliation. Extracting the data of a foliation of isochrons is achieved with `iscrn_read_foliation`. The function takes very similar arguments to `iscrn_read_isochron`, except that the isochron name must be formatted as in the example below. The following commands extract both sides of the foliation $\mathcal{I}(\Gamma^s)$ that is stored somewhere in MATLAB's search path, in a directory with the name format described in subsection SM1.6. Specifically, the computed isochrons of the foliation $\mathcal{I}(\Gamma^s)$ are read from the `po` directory, with sub-directories `iscrn%02d`, `arc%01d+` and `arc%01d-` storing the isochrons $I_\theta(\Gamma^s)$, and their arcs $I_\theta^k(\Gamma^s)$ for both sides, respectively.

```
>> GsP = iscrn_read_foliation('po/iscrn%02d/arc%01d+', 'T%d', {'x0_1', 'x0_2'},...
     '-trunc', 'initial');
>> GsM = iscrn_read_foliation('po/iscrn%02d/arc%01d+-', 'T%d', {'x0_1', 'x0_2'},...
     '-trunc', 'initial');
```

The output assigned to `GsP` and `GsM` are arrays of structures with fields similar to those of the structure `iscrn` from subsection SM1.6.2. Further details on the fields of these structures can be found in the help section of `iscrn_read_foliation`.

A foliation of forward-time isochrons is visualized by the following commands; note that each structure contained in the array `GsP` is accessed here with curved brackets rather than braces.

```
>> figure(1)
>> plot(GsP(1).xpo(:,1), GsP(1).xpo(:,2), 'k')
>> hold on
```

```
>> for i = 1:length(fol)
>>   plot(GsP(i).iscrn(:,1), GsP(i).iscrn(:,2),...
       'Color', iscrn_map_forward((i − 1) / length(GsP)))
>> end
>> xlabel('x')
>> ylabel('y', 'rotation', 0)
>> cmpf = iscrn_map_forward(0:1/100:0.99);
>> colormap(cmpf)
>> colorbar('Location', 'SouthOutside')
```

Each isochron is colored accordingly to its phase by the function `iscrn_map_forward`, which returns an rgb-value from the color-map used throughout this paper when supplied with a phase. A color map is easily constructed by passing a linear array of phases between 0 and 1 to the function, as in the third to last line above. MATLAB returns the left-hand plot in Figure SM2; compare with Figures 3 and 12.

The commands required to extract the foliation $\mathcal{U}(q_+)$ stored with the name format `'ep/iscrn%02d/arc%01d+'` are similar to those for periodic orbits, although the isochrons of equilibria have only one side.

```
>> qPP = iscrn_read_foliation('ep/iscrn%02d/arc%01d+', 'T%d', {'x0_1', 'x0_2'},...
     '−trunc', 'initial');
```

The output assigns to `qPP` an array of structures with the same fields as `GsP` above. This foliation of backward-time isochrons is visualized by the following commands.

```
>> figure(2)
>> plot(qPP(1).xpo(:,1), qPP(1).xpo(:,2), 'k')
>> hold on
>> for i = 1:length(qPP)
>>   plot(qPP(i).iscrn(:,1), qPP(i).iscrn(:,2),...
       'Color', iscrn_map_backward((i − 1) / length(qPP)))
>> end
>> xlabel('x')
>> ylabel('y', 'rotation', 0)
>> cmpb = iscrn_map_backward(0:1/100:0.99);
>> colormap(cmpb)
>> colorbar('Location', 'SouthOutside')
```

Again, each isochron is colored according to its phase, with rgb-values for backward-time phase returned by `iscrn_map_backward`. The resulting output is shown as the right-hand plot of Figure SM2; compare with Figures 3 and 12.

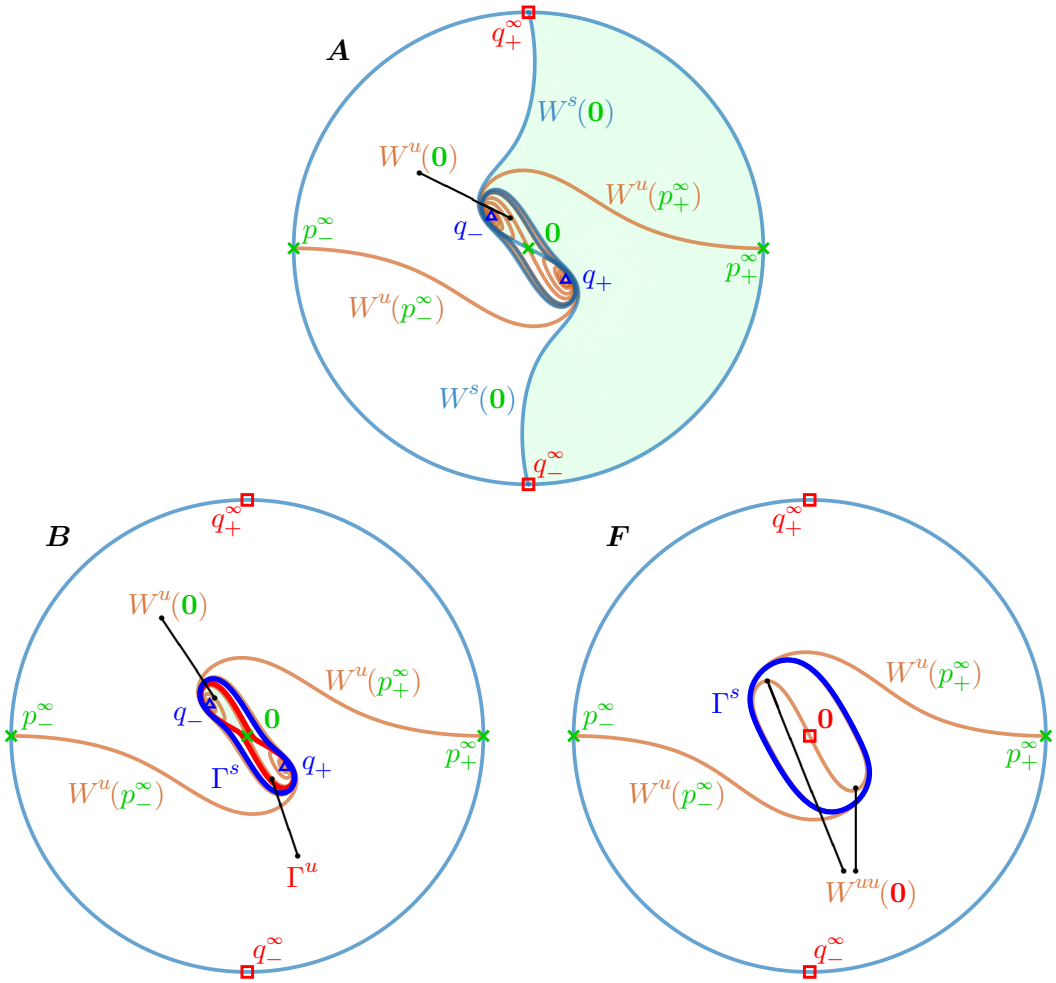

Figure SM3. *Compactified phase portraits of* (10) *in regions* $\boldsymbol{A}$, $\boldsymbol{B}$, *and* $\boldsymbol{F}$. *The unit circle* $\mathbb{S}^1$ *represents the direction of escape to infinity and consists two repellors* q^∞ (□), *two saddles* $p_\pm^\infty$ (×) *and their stable manifolds* $W^s(p_\pm^\infty)$ *(blue curves). Also shown are the unstable manifolds* $W^u(p_\pm^\infty)$ *(orange curves) and the respective finite invariant sets* $q_\pm$ (△) *and* $\mathbf{0}$ (×), *with manifolds* $W^s(\mathbf{0})$ *(light-blue curve),* $W^u(\mathbf{0})$ *(orange curve) and* $W^{uu}(\mathbf{0})$ *(orange curve), and the periodic orbit* Γ^s *(blue curve); compare with Figure* 2.

SM2. Dynamics at infinity. The phase portraits near infinity can be studied by compactifying phase space. This is achieved via Poincaré compactification [SM4, SM9], which maps $\mathbb{R}^2$ diffeomorphically to the open unit disc $\mathbb{D}^2$ such that its boundary $\mathbb{S}^1$ represents the direction of approach or departure from infinity; see also [SM5] for more details and explicit formulas. This coordinate change preserves the geometry of invariant objects and results in a continuous vector field defined on a (bounded) closed unit disc $\mathbb{D}^2 \cup \mathbb{S}^1$. Phase portraits in the compactification are equivalent to those in the original system, but with a non-linear rescaling of time; here, points of $\mathbb{S}^1$ correspond to direction of escape to infinity of the original system, which are determined by the higher-order terms.

Figure SM3 shows compactified phase portraits in regions $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{F}$. The invariant

objects on the unit circle $\mathbb{S}^1$ representing infinity are unaffected by the linear terms of (10). Since the bifurcation parameter a only appears in the linear terms of (10), the dynamics on $\mathbb{S}^1$ remain unchanged throughout regions $\boldsymbol{A}$–$\boldsymbol{F}$. On $\mathbb{S}^1$ there are two nodal repellors $q^\infty_\pm$ $(0, \pm 1)$ and two saddles $p^\infty_\pm$ at $(\pm 1, 0)$; their stable invariant manifolds $W^s(p^\infty_\pm)$ accumulate onto $q^\infty_\pm$ and trace out $\mathbb{S}^1$. The presence of the equilibria $q^\infty_\pm$ and $p^\infty_\pm$ on $\mathbb{S}^1$ mean that trajectories of (10) do not spiral towards infinity, but escape to infinity (in forward or backward time) along the directions represented by the points $q^\infty_\pm$ and $p^\infty_\pm$. Hence, the only isochron foliations that we need to concern ourselves with are those of finite invariant objects.

The compactified phase portrait of region $\boldsymbol{A}$ in Figure SM3 shows that the two branches of $W^s(\mathbf{0})$ accumulate in backward time to the repelling equilibria q^∞_- and q^∞_+, respectively. This shows that the curve $W^s(\mathbf{0})$, which forms the finite part of $\partial\mathcal{A}(q_\pm)$, winds around the equilibria $\mathbf{0}$ and $q_\pm$ only finitely many times. As a result, the (shaded) basin $\mathcal{A}(q_-)$ encompasses most of the right-hand side of the image; in particular, the semicircle $W^s(p^\infty_+)$ is also in the boundary $\partial\mathcal{A}(q_-)$, and the equivalent statement is true for $\mathcal{A}(q_-)$. Hence, the isochron foliations $\mathcal{I}(q_\pm)$ must accumulate onto both finite and non-finite invariant objects. Notice further that each basin $\mathcal{A}(q_\pm)$ contains the unstable manifold $W^u(p^\infty_\mp)$ which, hence, converges in forward time to $q_\pm$.

In the compactified phase portrait of region $\boldsymbol{B}$ in Figure SM3 the pair of periodic orbits Γ^s and Γ^u now surround $\mathbf{0}$ and $q_\pm$. Hence, $W^s(\mathbf{0})$ accumulates onto Γ^u in backward time and is no longer able to reach the points $q^\infty_\pm$, so that we now have a pair of winding basins of attraction $\mathcal{A}(q_\pm)$ inside of Γ^u. Similarly, the unstable manifolds $W^u(p^\infty_\pm)$ accumulate onto Γ^s and no longer reach the attractors $q_\mp$. Note that the basin $\mathcal{A}(\Gamma^s)$ outside Γ^s is bounded by the circle $\mathbb{S}^1$, that is, by the closure $\overline{W^s(p^\infty_-) \cup W^s(p^\infty_+)}$. The situation outside the stable periodic orbit Γ^s remains unchanged in the transitions through regions regions $\boldsymbol{C}$ to $\boldsymbol{F}$, as is illustrated in Figure SM3 with the compactified phase portrait of region $\boldsymbol{F}$. Hence, any qualitative changes of isochron geometry from region $\boldsymbol{B}$ onwards occur strictly inside Γ^s.

SM3. Table of isochron properties across the transition. We summarize the transitions from regions $\boldsymbol{A}$–$\boldsymbol{F}$ in a compact way in Table SM1 (continued over three pages) in terms of the properties of the invariant objects and the isochron foliations in the different basins. This information is very dense and will be very hard to digest on a stand-alone basis; rather this summary is provided as a look-up table that the reader may wish to consult in conjunction with the detailed explanations in section 3. Table SM1 is organized into subtables for each region, separated by subtables for each of the transitions between them. Each subtable for a region presents all equilibria and periodic orbits, the invariant manifolds of saddle equilibria, the basins of attractors/repellors and their boundaries, and the properties of the relevant isochron foliations; we then list the intersections of basins of attraction and repulsion in which isochron foliations coexist, and denote whether the forward-time and backward-time isochron foliations are transverse or have quadratic tangencies. Each subtable for a transition indicates the bifurcation in terms of the invariant objects that characterize it, and then details the associated changes to invariant objects, basins and isochron foliations.

Table SM1

*Summary of the phase portraits for regions **A**–**F** and the transitions between them. The subtables for each region present the respective label (first column), the invariant objects (second column), and their invariant manifolds for saddles and basins of attraction/repulsion, their boundaries and any isochron foliations (third column); we also indicate whether basins are unbounded or winding and list separately the intersection sets with interacting isochron foliations, where ⋔ denotes transverse and ⋔̸ tangent foliations. The subtables for each transitions between regions show the transition and bifurcating objects (first column) and then list the respective changes to the relevant invariant objects and foliations.*

A	$\mathbf{0}$	$W^s(\mathbf{0})$, $W^u(\mathbf{0})$	
	$p^\infty_\pm$	$W^s(p^\infty_\pm)$, $W^u(p^\infty_\pm)$	
	$q^\infty_\pm$	$\mathcal{R}(q^\infty_\pm)$ (unbounded)	
		$\partial\mathcal{R}(q^\infty_\pm) = q_- \cup q_+ \cup W^u(\mathbf{0}) \cup W^u_\pm(p^\infty_-) \cup W^u_\pm(p^\infty_+)$	
	$q_\pm$	$\mathcal{I}(q_\pm) \subset \mathcal{A}(q_\pm)$ (unbounded)	
		$\partial\mathcal{A}(q_\pm) = W^s(\mathbf{0}) \cup \overline{W^s(p^\infty_\mp)} = W^s(\mathbf{0}) \cup q^\infty_- \cup q^\infty_+ \cup W^s(p^\infty_\mp)$	
		$\mathcal{A}(q_-) \cap \mathcal{R}(q^\infty_\pm)$	$\mathcal{I}(q_-)$
		$\mathcal{A}(q_+) \cap \mathcal{R}(q^\infty_\pm)$	$\mathcal{I}(q_+)$
A $\leftrightarrow$ ***B***		$\emptyset \leftrightarrow \Gamma^s$ $\quad \emptyset \leftrightarrow \mathcal{I}(\Gamma^s)$	
SNL		$\emptyset \leftrightarrow \Gamma^u$ $\quad \emptyset \leftrightarrow \mathcal{U}(\Gamma^u)$	$\lim_{t\to-\infty} W^s(\mathbf{0}) : q^\infty_\pm \leftrightarrow \Gamma^u$
$\Gamma^s = \Gamma^u$		$\mathcal{A}(q_\pm)$: unbounded $\leftrightarrow$ winding basins	
		$\partial\mathcal{A}(q_\pm) : W^s(\mathbf{0}) \cup \overline{W^s(p^\infty_\mp)} \leftrightarrow \overline{W^s(\mathbf{0})}$	
B	$\mathbf{0}$	$W^s(\mathbf{0})$, $W^u(\mathbf{0})$	
	$q_\pm$	$\mathcal{I}(q_\pm) \subset \mathcal{A}(q_\pm)$ (winding basins)	
		$\partial\mathcal{A}(q_\pm) = \overline{W^s(\mathbf{0})} = W^s(\mathbf{0}) \cup \Gamma^u$	
	Γ^u	$\mathcal{U}(\Gamma^u) \subset \mathcal{R}(\Gamma^u)$	
		$\partial\mathcal{R}(\Gamma^u) = \overline{W^u(\mathbf{0})} \cup \Gamma^s = q_- \cup q_+ \cup W^u(\mathbf{0}) \cup \Gamma^s$	
	Γ^s	$\mathcal{I}(\Gamma^s) \subset \mathcal{A}(\Gamma^s)$	
		$\partial\mathcal{A}(\Gamma^s) = \Gamma^u \cup \infty = \Gamma^u \cup q^\infty_- \cup q^\infty_+ \cup W^s(p^\infty_-) \cup W^s(p^\infty_+)$	
		$\mathcal{A}(q_\pm) \cap \mathcal{R}(\Gamma^u)$	$\mathcal{I}(q_\pm) \not\pitchfork \mathcal{U}(\Gamma^u)$
		$\mathcal{A}(\Gamma^s) \cap \mathcal{R}(\Gamma^u)$	$\mathcal{I}(\Gamma^s) \pitchfork \mathcal{U}(\Gamma^u)$
B $\leftrightarrow$ ***C***		$\Gamma^u \leftrightarrow \emptyset$ $\quad \mathcal{U}(\Gamma^u) \leftrightarrow \emptyset$	$\lim_{t\to\infty} W^u(\mathbf{0}) : q_\pm \leftrightarrow \Gamma^s$
HOM		$\emptyset \leftrightarrow \Gamma^u_\pm$ $\quad \emptyset \leftrightarrow \mathcal{U}(\Gamma^u_\pm)$	$\lim_{t\to-\infty} W^s(\mathbf{0}) : \Gamma^u \leftrightarrow \Gamma^u_\pm$
$W^s(\mathbf{0}) = W^u(\mathbf{0})$		$\mathcal{A}(q_\pm)$: winding basins $\leftrightarrow$ disks	
		$\partial\mathcal{A}(q_\pm) : \overline{W^s(\mathbf{0})} \leftrightarrow \Gamma^u_\pm$	
		$\partial\mathcal{A}(\Gamma^s) : \Gamma^u \cup \infty \leftrightarrow \overline{W^s(\mathbf{0})} \cup \infty$	

Table SM1
continued.

C	$\mathbf{0}$	$W^s(\mathbf{0})$, $W^u(\mathbf{0})$		
	$q_\pm$	$\mathcal{I}(q_\pm) \subset \mathcal{A}(q_\pm)$		
		$\partial\mathcal{A}(q_\pm) = \Gamma^u_\pm$		
	$\Gamma^u_\pm$	$\mathcal{U}(\Gamma^u_\pm) \subset \mathcal{R}(\Gamma^u_\pm)$ (winding basins)		
		$\partial\mathcal{R}(\Gamma^u_\pm) = q_\pm \cup \overline{W^u(\mathbf{0})} = q_\pm \cup W^u(\mathbf{0}) \cup \Gamma^s$		
	Γ^s	$\mathcal{I}(\Gamma^s) \subset \mathcal{A}(\Gamma^s)$		
		$\partial\mathcal{A}(\Gamma^s) = \overline{W^s(\mathbf{0})} \cup \infty = W^s(\mathbf{0}) \cup \Gamma^u_- \cup \Gamma^u_+ \cup \infty$		
		$\mathcal{A}(q_\pm) \cap \mathcal{R}(\Gamma^u_\pm)$	$\mathcal{I}(q_\pm) \pitchfork \mathcal{U}(\Gamma^u_\pm)$	
		$\mathcal{A}(\Gamma^s) \cap \mathcal{R}(\Gamma^u_\pm)$	$\mathcal{I}(\Gamma^s) \not\pitchfork \mathcal{U}(\Gamma^u_\pm)$	
C $\leftrightarrow$ ***D***		$\Gamma^u_\pm \leftrightarrow \emptyset$	$\mathcal{U}(\Gamma^u_\pm) \leftrightarrow \emptyset$	$\lim_{t\to-\infty} W^s(\mathbf{0}) : \Gamma^u_\pm \leftrightarrow q_\pm$
H		$\mathcal{A}(q_\pm) \leftrightarrow \emptyset$	$\mathcal{I}(q_\pm) \leftrightarrow \emptyset$	$q_\pm$: attractor $\leftrightarrow$ repellor
hyperbolic $q_\pm$		$\mathcal{R}(q_\pm) \leftrightarrow \emptyset$	$\emptyset \leftrightarrow \mathcal{U}(q_\pm)$	
D	$\mathbf{0}$	$W^s(\mathbf{0})$, $W^u(\mathbf{0})$		
	$q_\pm$	$\mathcal{U}(q_\pm) \subset \mathcal{R}(q_\pm)$ (winding basins)		
		$\partial\mathcal{R}(q_\pm) = \overline{W^u(\mathbf{0})} = W^u(\mathbf{0}) \cup \Gamma^s$		
	Γ^s	$\mathcal{I}(\Gamma^s) \subset \mathcal{A}(\Gamma^s)$		
		$\partial\mathcal{A}(\Gamma^s) = \overline{W^s(\mathbf{0})} \cup \infty = q_- \cup q_+ \cup W^s(\mathbf{0}) \cup \infty$		
		$\mathcal{A}(\Gamma^s) \cap \mathcal{R}(q_\pm)$	$\mathcal{I}(\Gamma^s) \not\pitchfork \mathcal{U}(q_\pm)$	
D $\leftrightarrow$ ***E***		$\mathcal{I}(q_\pm) \leftrightarrow \emptyset$		$q_\pm$: focus $\leftrightarrow$ node
CC				
$q_\pm$: repeated eigenvalue and eigenvector				
E	$\mathbf{0}$	$W^s(\mathbf{0})$, $W^u(\mathbf{0})$		
	$q_\pm$	$\mathcal{R}(q_\pm)$ (winding basins)		
		$\partial\mathcal{R}(q_\pm) = \overline{W^u(\mathbf{0})} = W^u(\mathbf{0}) \cup \Gamma^s$		
	Γ^s	$\mathcal{I}(\Gamma^s) \subset \mathcal{A}(\Gamma^s)$		
		$\partial\mathcal{A}(\Gamma^s) = \overline{W^s(\mathbf{0})} \cup \infty = q_- \cup q_+ \cup W^s(\mathbf{0}) \cup \infty$		
		$\mathcal{A}(\Gamma^s) \cap \mathcal{R}(q_\pm)$	$\mathcal{I}(\Gamma^s)$	

Table SM1
continued.

$\boldsymbol{E} \leftrightarrow \boldsymbol{F}$	$q_\pm \leftrightarrow \emptyset$	$\mathcal{R}(q_\pm) \leftrightarrow \emptyset$	$\mathbf{0}$: saddle $\leftrightarrow$ repellor
PF		$\emptyset \leftrightarrow \mathcal{R}(\mathbf{0})$	$W^u(\mathbf{0}) \leftrightarrow W^{uu}(\mathbf{0})$
$\mathbf{0} = q_\pm$			

$\boldsymbol{F}$	$\mathbf{0}$		
		$\mathcal{R}(\mathbf{0})$, $W^{uu}(\mathbf{0})$	
		$\partial\mathcal{R}(\mathbf{0}) = \Gamma^s$	
	Γ^s	$\mathcal{I}(\Gamma^s) \subset \mathcal{A}(\boldsymbol{\Gamma^s})$	
		$\partial\mathcal{A}(\boldsymbol{\Gamma^s}) = \mathbf{0} \cup \infty$	
		$\mathcal{A}(\boldsymbol{\Gamma^s}) \cap \mathcal{R}(\mathbf{0})$	$\mathcal{I}(\Gamma^s)$

REFERENCES

[1] H. Dankowicz and F. Schilder, *CoCo: General-purpose tools for continuation and bifurcation analysis of dynamical systems*, 2009, https://sourceforge.net/projects/cocotools/.

[2] H. Dankowicz and F. Schilder, *Recipes for Continuation*, Computational Science & Engineering, Society for Industrial and Applied Mathematics, Philadelphia, 2013, https://doi.org/10.1137/1.9781611972573.

[3] E. J. Doedel, B. E. Oldeman, A. R. Champneys, F. Dercole, T. F. Fairgrieve, Y. A. Kuznetsov, Paffenroth, B. Sandstede, X. Wang, and C. Zhang, *AUTO-07P: Continuation and bifurcation software for ordinary differential equations*, tech. report, Concordia University, Montreal, Canada, 2012, http://cmvl.cs.concordia.ca/auto/.

[4] E. A. González-Velasco, *Generic properties of polynomial vector fields at infinity*, Transactions of the American Mathematical Society, 143 (1969), pp. 201–222, https://doi.org/10.1090/S0002-9947-1969-0252788-8.

[5] J. Hannam, B. Krauskopf, and H. M. Osinga, *Global isochrons of a planar system near a phaseless set with saddle equilibria*, The European Physical Journal Special Topics, 225 (2016), pp. 2645–2654, https://doi.org/10.1140/epjst/e2016-60072-4.

[6] B. Krauskopf and T. Riess, *A Lin's method approach to finding and continuing heteroclinic connections involving periodic orbits*, Nonlinearity, 21 (2008), pp. 1655–1690, https://doi.org/10.1088/0951-7715/21/8/001.

[7] P. Langfield, B. Krauskopf, and H. M. Osinga, *Solving Winfree's puzzle: The isochrons in the FitzHugh–Nagumo model*, Chaos: An Interdisciplinary Journal of Nonlinear Science, 24 (2014), p. 013131, https://doi.org/10.1063/1.4867877.

[8] P. Langfield, B. Krauskopf, and H. M. Osinga, *Forward-time and backward-time isochrons and their interactions*, SIAM Journal on Applied Dynamical Systems, 14 (2015), pp. 1418–1453, https://doi.org/10.1137/15M1010191.

[9] M. Messias, *Dynamics at infinity and the existence of singularly degenerate heteroclinic cycles in the Lorenz system*, Journal of Physics A: Mathematical and Theoretical, 42 (2009), p. 115101, https://doi.org/10.1088/1751-8113/42/11/115101.

[10] H. M. Osinga and J. Moehlis, *Continuation-based computation of global isochrons*, SIAM Journal on Applied Dynamical Systems, 9 (2010), pp. 1201–1228, https://doi.org/10.1137/090777244.

[11] I. The MathWorks, *Formatting text*, 2020, https://www.mathworks.com/help/matlab/matlab_prog/formatting-strings.html.