Anastasiia Panchuk

Finding periodic points and invariant manifolds for maps A set of Matlab routines

User guide (ver. 0.9)

September 21, 2013

Contents

1 Initialization

Before starting to use the routine one should load the file `init.m` from the routine's root directory. This file will add the necessary directory paths to the Matlab's `PATH` variable, and will also load the set of method parameters with their default values. Namely, the following variables will be loaded:

Variable	Value	Description
`divergence`	`100000`	limit for detecting divergence to infinity (see Sec. 3.1)
`tolerance`	`1e-8`	accuracy for checking parity of points (see Secs. 4, 5)
`max_step`	`50`	maximal number of Newton steps (see Sec. 4)
`arc_length`	`10`	desired length of the manifold arc (see Sec. 5)
`d`	`0.001`	initial step for manifold calculation (see Sec. 5)
`a_min`	`0.2`	search circle minimal angle value (see Sec. 5)
`a_max`	`0.3`	search circle maximal angle value (see Sec. 5)
`Da_min`	`1e-6`	minimal value for the product $\Delta_k\alpha$ (see Sec. 5)
`Da_max`	`1e-5`	maximal value for the product $\Delta_k\alpha$ (see Sec. 5)
`D_min`	`1e-4`	search circle minimal step size (see Sec. 5)
`cB`	`0.2`	bisection accuracy coefficient (see Sec. 5.1)
`eB`	`1e-6`	search circle bisection error (see Sec. 5.2)

One is free to change the default method parameter values inside `init.m` adjusting them to one's own needs, as well as to add different directory paths to the Matlab's `PATH` variable if required.

The default method parameter values are also stored in the file `data/default.mat`.

2 New map definition

To start using the routine one would possibly like to define one's own map to study. For that it is enough to create a new Matlab script file with the appropriate function, typical view of which is presented in Listing 1 (see also Listing 2, other predefined sample files in the routine's directory `map/`, and Matlab package documentation [1] for more details).

Listing 1: user_map.m

```
function y = <user_map>(x, param1, param2, ....)
% a comment and description
{ some optional commands }
y(1) = <equation 1>
y(2) = <equation 2>
```

Listing 2: henon.m

```
function y = henon(x, a, b)
% Henon map
if (length(x) ~= 2)
    cprintf('err', 'ERROR: First argument must be ....');
    y = [];
    return;
else
    y(1) = x(2) + 1 - a*x(1)^2;
    y(2) = b*x(1);
end
```

A Jacobi matrix for the user defined map may be given explicitly as well, which appears to be useful while finding periodic points (with the Newton method) and calculating their eigenvalues and eigenvectors. **Important:** note, that the function for the user defined Jacobi matrix *must* be of particular specification (see Listing 3):

- its name is of the form `<user_map>_J`, where `<user_map>` is the name of the corresponding user defined map,
- it has *the same* set of parameters as the fucntion for the corresponding user defined map `<user_map>`.

Listing 3: henon_J.m

```
function J = henon_J(x, a, b)
% Jacobi matrix for the Henon map
if (length(x) ~= 2)
    cprintf('err', 'ERROR: First argument must be ....');
    y = [];
    return;
else
    J = zeros(2, 2);
    J(1, 1) = -2*a*x(1);
    J(1, 2) = 1;
    J(2, 1) = b;
end;
```

3 Period diagrams

The procedure to produce 2D bifurcation diagrams (period diagrams) consists usually of two steps. First, one obtains the data for the diagram, and then uses this data to make the 2D colour plot. The method for getting the period data is rather straightforward: starting from the given initial point a target map is iterated the certain number of times, and then the last obtained point is checked for being periodic (with the period being not greater than the given maximal value).

3.1 Obtaining period data

To obtain the period data for a 2D bifurcation diagram one should use the function `period_bd_calc` defined in `main/period_bd_calc.m`.

Listing 4: period_bd_calc.m

```
function [param1, param2, periods] = ...
          period_bd_calc(fhandle, x0, prange, dp, ...
          pidx, trans, maxperiod, tolerance, ...
          divergence, ....)
%
% Calculate the data for the 2D period diagram
....
```

Parameters. The function requires the following parameters:

`fhandle` A function handle corresponding to the desired map, for instance `@henon` for the Hénon map presented in Listing 2 (for more details on function handles see [1]).

`x0` An initial point for obtaining the trajectory. Namely, for each parameter pair the initial value is always reset to `x0` if it is a numeric array. However, one may force the initial condition to be chosen randomly at each cycle turn by specifying `x0` as a string of the form `'rand(<n>)'` where `<n>` is the required intial vector dimension (see Listing 5).

`prange` A parameter range which is a 2×2 matrix with the i-th row corresponding to the i-th parameter ($i = 1, 2$). Namely, if the first parameter changes from 0 to 2 and the second one from -3 to 3, then `prange` should be `[0 2; -3 3]`.

`dp` A 2-dimensional vector defining the parameter increments (*e. g.*, `[0.05 0.01]`).

`pidx` A 2-dimensional vector defining the parameter indices. In the map definition the parameters always follow in the certain order, for instance, in `function y = skewtent(x, a, b, mu)` the first parameter is always `a`, the second is `b`, and the third one is `mu`. For the diagram in the `(a, b)`-plane the parameter `pidx` is `[1 2]`, while for the one in the `(a, mu)`-plane it is `[1 3]`.

`trans` A transient number of iterations which are skipped before the orbit is checked for being periodic or not.

`maxperiod` A maximal period to search.

`tolerance` An accuracy for checking parity of points, namely, if the distance (norm) between the two points x_1 and x_2 is less than `tolerance`, then these points are considered to be equal.

`divergence` A limit for detecting divergence to infinity, namely, if the radius (norm) of the orbit point x is larger than `divergence`, then the orbit is considered to go to infinity.

`....` The parameters of the iterated map, which may be specified in two ways. They can be given either as a simple comma separated list, or as a single cell-array parameter (see Listing 5, note that `{:}` is important). It should be mentioned, that for the two target parameters (which will be then varied for obtaning the diagram), one should give their values here as well, although these values will not be used.

Return values. The return values of the function are the following:

`param1` A vector with the mesh in the first parameter

`param2` A vector with the mesh in the second parameter

`periods` A matrix of the period values related to the resulting parameter mesh. Note, that the value `-1` corresponds to the orbits being divergent to infinity, and `0` is related to either the case of non-regular (*e. g.*, chaotic or strange) behaviour, or to the orbits of period larger than `maxperiod`.

The return values of `period_bd_calc` can be directly used in `bd_plot` for plotting the bifurcation diagram. The further example of the function usage may be found in `do_bd_sample.m`.

Listing 5: do_bd_sample.m

```
% the skewtent map, predefined initial condition
[param1, param2, periods] = ...
  period_bd_calc(@skewtent, 0.35, [0 3; -10 0], ...
  [0.02 0.05], [1 2], 2000, 15, 1e-8, 100000, 1, -1, 1);
....
% the modified Ikeda map, predefined initial condition
param = {1, 0.9, 1, 0.4, 6};
% pay attention to the last parameter param{:}
% note, that {:} is important here
[param1, param2, periods] = ...
  period_bd_calc(@ikeda, [1 2], [0.5 1.5; 0.5 1.5], ...
  [0.05 0.05], [1 2], 2000, 15, 1e-8, 100000, param{:});
....
% the bimodal PWL map, random intial condition
[param1, param2, periods] = ...
  period_bd_calc(@pwl3part, 'rand(1)', [-1 15; -15 1],
     ...
  [0.2 0.2], [3 4], 2000, 30, 1e-8, 100000, ...
  0.85, 0.8, 1, -1, 0, 0.3);
....
```

Displayed output. While running, the function displays some information about the data calculation progress in the Command Window. At each cycle turn, when the first parameter changes its current value is displayed in orange, and the approximate time left is displayed in blue. (For colorizing the output the utility `cprintf` is used, see [2]).

3.2 Plotting period diagram

To plot a 2D bifurcation diagram (period diagram) from the data obtained by `period_bd_calc` one should use `bd_plot` defined in `plot/bd_plot.m`.

Listing 6: bd_plot.m

```
function [hfig, hcbar] = bd_plot(X, Y, C, maxperiod)
%
% Plot colour-coded period diagram
....
```

Parameters. The function requires the following parameters:

`X`	A vector of horizontal axes mesh (one parameter)
`Y`	A vector of vertical axes mesh (the other parameter)
`C`	A matrix of the period values related to the given parameter mesh
`maxperiod`	maximal period to plot

The plot will be produced by using the preset colour palette (defined in `data/bd_colors.mat`). It is limited up to maximum of 62 colours, where grey colour corresponds to the value `-1` (divergence), and white colour is related to `0` (higher periodic or non-regular behaviour). Note, that if the matrix `C` contains cells with values being larger than `maxperiod`, then they will be also coloured white in the produced plot. The return values can be used for further customization of the figure. (See Listing 7 and `example/do_bd_sample.m`).

Return values. The function returns the following values:

`hfig`	A handle for the plotted figure window
`hcbar`	A handle for the plotted colour bar

Listing 7: do_bd_sample.m

```
% plot BD data with the same max period as was calculated
bd_plot(param1, param2, periods, 30);
....
% plot BD data only up to periods 7
[hfig, hcbar] = bd_plot(param1, param2, periods, 7);
% further customizing created figure
% change figure size and position
set(hfig, 'Units', 'centimeters');
set(hfig, 'Position', [2, 2, 18, 14]);
....
% change colorbar range and ticks
set(hcbar, 'ylim', [-1.5 16.5], 'YTick', (0:2:16));
```

4 Find periodic points

In order to find periodic points the Newton-Raphson method for approximating zero-roots of the function is used (see, *e. g.*, [3, §9.6]). For multidimensional maps certain modifications improving the method convergence are added (*ibid.* §9.7).

4.1 One-dimensional maps

The method for 1D maps is implemented in the function `find_periodic_1d` (see the file `main/find_periodic_1d.m`).

Listing 8: find_periodic_1d.m

```
function xF = find_periodic_1d(fhandle, period, ...
  interval, init_num, tolerance, max_step, rnd, ....)
%
% Finding periodic points of the specified period
....
```

Parameters. The function requires the following parameters:

`fhandle`	A function handle corresponding to the desired map (for more details on function handles see [1]).
`period`	A period of which the points are searched. Note, that the points with periods being divisors of `period` are also may be found. For instance, if `period` is `10`, in general, the result may contain `1`-, `2`-, `5`-, and `10`-periodic points.
`interval`	Usually, defines a state variable interval from which the initial conditions are taken, but if `init_num` is zero, then `interval` is considered to be a predefined initial value.
`init_num`	The number of initial conditions to try. If `init_num` is zero, the initial condition is read from the parameter `interval`.
`tolerance`	An accuracy for checking parity of points, namely, if the distance (norm) between the two points x_1 and x_2 is less than `tolerance`, then these points are considered to be equal.

`max_step` maximal number of Newton steps performed.

`rnd` Set initial conditions randomly or not. If `true` (or `1`), then initial conditions are chosen randomly (in the total amount of `init_num`). If `false` (or `0`), then the search interval is divided into `init_num` equal parts, and then a middle point of each subinterval is taken for the initial condition.

`....` The parameters of the iterated map, which may be specified in two ways. They can be given either as a simple comma separated list, or as a single cell-array parameter (see Listing 9 and `example/do_points_sample.m`, note that `{:}` is important).

Return value. The function returns the following values:

`xF` An array of the found (distinct) points sorted in acsending order.

Listing 9: do_points_sample.m

```
....
xF = find_periodic_1d(@logistic, 10, [0 1], 100, ...
     1e-8, 50, true, 3.8);
....
param = {0.47, -9, 1};
xF = find_periodic_1d(@skewtent, 12, [-10 10], 100, ...
     1e-8, 50, true, param{:});
```

Displayed output. While running, the function displays some information about the calculation progress in the Command Window. Each run (initial condition tried) number appears in orange, the total number of Newton steps is displayed in blue, and if the Newton method fails to converge a warning is displayed in green. The final statistics about how many distinct periodic points were found is printed in violet. (For colorizing the output the utility `cprintf` is used [2]).

4.2 Multi-dimensional maps

For finding the periodic points in multidimensional maps the function `find_periodic_md` should be used (see the file `main/find_periodic_md.m`).

Listing 10: find_periodic_md.m

```
function xF = find_periodic_md(fhandle, period, ...
range, init_num, tolerance, max_step, rnd, ....)
%
% Find periodic points of the function
% multi-dimensional
....
```

Parameters. The function requires the following parameters:

`fhandle`	A function handle corresponding to the desired map (for more details on function handles see [1]).
`period`	A period of which the points are searched. Note, that the points with periods being divisors of `period` are also may be found. For instance, if `period` is `10`, in general, the result may contain `1-`, `2-`, `5-`, and `10-`periodic points.
`range`	Usually, defines a state variable interval from which the initial conditions are taken, but if `init_num` is zero, then `range` is considered to be a predefined initial value.
`init_num`	The vector indicating the number of subdivisions for each coordinate. Namely, the i-th element of `init_num` is the number of pieces into which the i-th interval from `range` will be divided. If the length `init_num` is less than the map dimension, the missing elements are put to be 1. Note that if `init_num` is zero, the initial condition is read directly from the parameter `range` (see Listing 11).
`tolerance`	An accuracy for checking parity of points, namely, if the distance (norm) between the two points x_1 and x_2 is less than `tolerance`, then these points are considered to be equal.
`max_step`	A maximal number of Newton steps performed.

`rnd` Set initial conditions randomly or not. If `true` (or `1`), then initial conditions are chosen randomly (in the total amount of `init_num`, which should be a single positive integer in this case). If `false` (or `0`), then

- either the parameter `range` is used as a predefined intial condition (if `init_num` is `0`);
- or the search range (given in `range`) is divided into subdomains (n-dimensional *parallelepipeds* or *parallelotopes*) according to the integer vector given in `init_num` (see also the description of the `init_num` parameter), and then a center point of each such subdomain is taken for the initial condition.

(see Listing 11 and `example/do_points_sample.m`)

`...` The parameters of the iterated map, which may be specified in two ways. They can be given either as a simple comma separated list, or as a single cell-array parameter (see Listing 11 and `example/do_points_sample.m`, note that `{:}` is important).

Return value. The function returns the following values:

`xF` An array of the found (distinct) points sorted in ascending order by the first coordinate.

Displayed output. While running, the function displays some information about the calculation progress in the Command Window. Each run (initial condition tried) number appears in orange, the total number of Newton steps is displayed in blue, and if the Newton method fails to converge a warning is displayed in green. The final statistics about how many distinct periodic points were found is printed in violet. (For colorizing the output the utility `cprintf` is used [2]).

Listing 11: do_points_sample.m

```
....
param = {1, 0.9, 1, 0.4, 6};
% random initial conditions
xF = find_periodic_md(@ikeda, 2, [-5 5; -5 5], 200, ...
     1e-8, 50, true, param{:});
% uniformly distributed initial conditions
xF = find_periodic_md(@ikeda, 2, [-5 5; -5 5], [10 10],
    ...
     1e-8, 50, false, param{:});
% given initial condition
xF = find_periodic_md(@ikeda, 2, [0.5 0.2], 0, ...
     1e-8, 50, false, param{:});
```

4.3 Classifying found periodic points

As the functions `find_periodic_1d` and `find_periodic_md` return only a sorted array of points, neither indicating exact periods of them, nor giving the information about their stability, one may need to classify this output, and check whether some of the points belong to the same cycle. For this, one should use `period_classify` (defined in `main/period_classify.m`).

Listing 12: period_classify.m

```
function points = period_classify(fhandle, xF, ...
  max_period, tolerance, ....)
```

Parameters. The function requires the following parameters:

`fhandle`	A function handle corresponding to the desired map (for more details on function handles see [1]).
`xF`	A given array of points, which are to be classified.
`max_period`	A maximal point period to be checked.
`tolerance`	An accuracy for checking parity of points, namely, if the distance (norm) between the two points x_1 and x_2 is less than `tolerance`, then these points are considered to be equal.

`....` The parameters of the iterated map, which may be specified in two ways. They can be given either as a simple comma separated list, or as a single cell-array parameter.

Return value. The function returns the following values:

`points` A cell-array of the classified cycles and fixed points. It has the following structure: each row has three cells, where the first cell contains the cycle (or a fixed point) coordinates, the second cell is the exact period of the cycle, and the third one indicates the stability—`'stable'`, `'unstable'`, or `'saddle'` (see Listing 13).

Listing 13: Sample output of `period_classify`

```
>> xF = find_periodic_1d(@logistic, 10, [0 1], 100, ...
        1e-8, 50, true, 3.8)
....
xF =
  -0.000000000000004
  0.185626749794394
  ....
  0.948498445176702
>> points = period_classify(@logistic, xF, 10, 1e-6, 3.8)
points =
    [-3.876193763738351e-15]    [ 1]    'saddle'
                [10x1 double]    [10]    'saddle'
                [ 5x1 double]    [ 5]    'saddle'
    ....
                [10x1 double]    [10]    'saddle'
    [     0.736842105262070]    [ 1]    'saddle'
```

5 Invariant manifolds

5.1 Unstable manifolds

For computing unstable manifolds the Search circle method is used (see, *e. g.*, [4]). In this method the manifold is grown iteratively with piecewise linear approximation. At each step the cirlce is plotted, with the center

in the last calculated point p_k and the radius Δ_k (the adoptive step value which may vary through the procedure). Further, it is assumed that the new manifold point p_{k+1} lies on this circle, and there exists a point p^* belonging to a certain (already computed) segment (p_{i-1}, p_i) which is mapped to p_{k+1} under the action of the iterated map. Then, for searching the correct p^* the bisection method is used. The resulting manifold approximation consists of a number of line segments (p_k, p_{k+1}).

5.1.1 Fixed points

To compute the unstable manifold of a fixed point one should use the function `sc_unstable` defined in `main/sc_unstable.m`.

Listing 14: sc_unstable.m

```
function [p, arc_length] = sc_unstable(fhandle, p0, ...
  arc_max, d, a_min, a_max, Da_min, Da_max, D_min, ...
  eB, ....)
%
% Find a 1D unstable manifold of a saddle fixed point
....
```

Parameters. The function requires the following parameters:

`fhandle`	A function handle corresponding to the desired map (for more details on function handles see [1]).
`p0`	A saddle point whose manifold is to be computed.
`arc_max`	A desired length of the manifold arc (is approximated as a sum of vector norms $\|p_{k+1} - p_k\|$, where $\{p_k\}_{k=0}^{N}$ are computed points of the manifold)
`d`	An initial step size. Note that if `d` > 0 manifold is grown forward, while if `d` < 0 it is grown backward.
`a_min`	A minimal angle value. If the angle α between the vectors (p_{k-1}, p_k) and (p_k, p_{k+1}) falls below this value, the step size is increased.

`a_max` A maximal angle value. The angle α between the vectors (p_{k-1}, p_k) and (p_k, p_{k+1}) is assumed to not exceed this threshold (for exception cases see [4]).

`Da_min` A minimal value for the product $\Delta_k \alpha$, where Δ_k is the current step size and α is the angle between (p_{k-1}, p_k) and (p_k, p_{k+1}).

`Da_max` A maximal value for the product $\Delta_k \alpha$, where Δ_k is the current step size and α is the angle between (p_{k-1}, p_k) and (p_k, p_{k+1}).

`D_min` A minimal step size.

`cB` A bisection accuracy coefficient, must be positive and less than one. If the distance between the point $\hat{p}_{k+1}$ (where $\hat{p}_{k+1}$ is a candidate for the next manifold point p_{k+1}) and the mentioned circle is less than `cB` $\cdot$ `R`, where `R` is the current circle radius, the point p_{k+1} is assumed to be found and the bisection stops.

`....` The parameters of the iterated map, which may be specified in two ways. They can be given either as a simple comma separated list, or as a single cell-array parameter (see *example?*, note that `{:}` is important).

To understand more deeply the role of the parameters `a_min`, `a_max`, `Da_min`, `Da_max`, `D_min`, and `cB` one is encouraged to refer to [4] and other works of the same authors.

Return values. The function returns the following values:

`p` An array of the computed manifold points.

`arc_length` A computed manifold arc length

Displayed output. While running, the function displays some output in the Command Window. Information about the calculation progress is printed in blue, warnings (*e. g.*, about inability to find the next point, or acceptance the point not satisfying the required constraints—for details see [4]) appear in green. Certain auxiliary information is displayed in orange, and the final data is printed in violet. (For colorizing the output the utility `cprintf` is

used [2]).

The example for usage of `sc_unstable` is presented in Listing 16.

5.1.2 Periodic points

To compute the unstable manifold of a periodic point with period greater than one, the function `sc_unstable_cyc` should be used (`sc_unstable_cyc.m`).

Listing 15: sc_unstable_cyc.m

```
function [p, arc_length] = sc_unstable_cyc(fhandle, ...
  p0, period, arc_max, d, a_min, a_max, Da_min, ...
  Da_max, D_min, eB, ....)
%
% Find a 1D unstable manifold of a saddle point
....
```

Parameters. The function requires the following parameters:

`fhandle`	A function handle corresponding to the desired map (for more details on function handles see [1]).
`p0`	A saddle point whose manifold is to be computed.
`period`	A period of the target saddle point.
`arc_max`	A desired length of the manifold arc (is approximated as a sum of vector norms $\|p_{k+1}-p_k\|$, where $\{p_k\}_{k=0}^{N}$ are computed points of the manifold)
`d`	An initial step size. Note that if `d` > 0 manifold is grown forward, while if `d` < 0 it is grown backward.
`a_min`	A minimal angle value. If the angle α between the vectors (p_{k-1}, p_k) and (p_k, p_{k+1}) falls below this value, the step size is increased (see [4]).
`a_max`	A maximal angle value. The angle α between the vectors (p_{k-1}, p_k) and (p_k, p_{k+1}) is assumed to not exceed this threshold (for exceptions see [4]).

`Da_min` A minimal value for the product $\Delta_k\alpha$, where Δ_k is the current step size and α is the angle between (p_{k-1}, p_k) and (p_k, p_{k+1}) (see [4]).

`Da_max` A maximal value for the product $\Delta_k\alpha$, where Δ_k is the current step size and α is the angle between (p_{k-1}, p_k) and (p_k, p_{k+1}) (see [4]).

`D_min` A minimal step size.

`cB` A bisection accuracy coefficient, must be positive and less than one. If the distance between the point $\hat{p}_{k+1}$ (where $\hat{p}_{k+1}$ is a candidate for the next manifold point p_{k+1}) and the mentioned circle is less than `cB` $\cdot$ `R`, where `R` is the current circle radius, the point p_{k+1} is assumed to be found and the bisection stops.

`....` The parameters of the iterated map, which may be specified in two ways. They can be given either as a simple comma separated list, or as a single cell-array parameter (see *example?*, note that `{:}` is important).

Return values. The function returns the following values:

`p` An array of the computed manifold points.

`arc_length` A computed manifold arc length

Displayed output. While running, the function displays some output in the Command Window. Information about the calculation progress is printed in blue, warnings (*e. g.*, about inability to find the next point, or acceptance the point not satisfying the required constraints—for details see [4]) appear in green. Certain auxiliary information is displayed in orange, and the final data is printed in violet. (For colorizing the output the utility `cprintf` is used [2]).

The example for usage of `sc_unstable_cyc` is presented in Listing 16.

Listing 16: do_unstab_sample.m

```
% modified Ikeda map
param = {1, 0.9, 1, 0.4, 6};
p0 = [1.08331887404 -2.4079634834];
[uf1, arc] = sc_unstable(@ikeda, p0, 10, 0.001, 0.2, ...
             0.3, 1e-6, 1e-5, 1e-4, 0.2, param{:});
[ub1, arc] = sc_unstable(@ikeda, p0, 10, -0.001, 0.2, ...
             0.3, 1e-6, 1e-5, 1e-4, 0.2, param{:});
....
% shallow map, period-2 point
p0 = [2.13884578746 -0.46130133276];
[uf, arc] = sc_unstable_cyc(@shallow, p0, 2, 30, 0.001,
    ...
            0.2, 0.3, 1e-6, 1e-5, 1e-4, 0.2, param{:});
[ub, arc] = sc_unstable_cyc(@shallow, p0, 2, 30, -0.001,
    ...
            0.2, 0.3, 1e-6, 1e-5, 1e-4, 0.2, param{:});
```

5.2 Stable manifolds

For computing stable manifolds the Search circle method is used (see [5]) The main advantage of the method is that the inverse is not needed. As well as in the case of unstable manifold, the stable manifold is grown iteratively with piecewise linear approximation. At each step the cirlce is plotted, with the center in the last calculated point p_k and the radius Δ_k (the adoptive step value which may vary through the procedure). Again it is assumed that the new manifold point p_{k+1} lies on this circle, and there exists a point p^* belonging to a certain (already computed) segment (p_{i-1}, p_i) which is an image of p_{k+1}. For searching p_{k+1} the bisection method is used as well. The resulting manifold approximation consists of a number of line segments (p_k, p_{k+1}).

5.2.1 Fixed points

To compute the stable manifold of a fixed point one should use the function `sc_stable` defined in `main/sc_stable.m`.

Listing 17: sc_unstable.m

```
function [p, arc_length] = sc_stable(fhandle, p0, ...
  arc_max, max_iter, d, a_min, a_max, Da_min, Da_max, ...
  D_min, eB, ....)
%
% Find a 1D stable manifold of a saddle fixed point
....
```

Parameters. The function requires the following parameters:

`fhandle` A function handle corresponding to the desired map (for more details on function handles see [1]).

`p0` A saddle point whose manifold is to be computed.

`arc_max` A desired length of the manifold arc (is approximated as a sum of vector norms $\|p_{k+1} - p_k\|$, where $\{p_k\}_{k=0}^{N}$ are computed points of the manifold)

`max_iter` A maximal number of iterates to try. This parameter is useful in case of a non-invertible map, which may have several preimages. The stable manifold of such a map may be mapped onto itself in a compex way, so that it is required to iterate the map several times until getting the right approximation for the next point (see [5, Example 4.3]).

`d` An initial step size. Note that if `d` > 0 manifold is grown forward, while if `d` < 0 it is grown backward.

`a_min` A minimal angle value. If the angle α between the vectors (p_{k-1}, p_k) and (p_k, p_{k+1}) falls below this value, the step size is increased (see [5]).

`a_max` A maximal angle value. The angle α between the vectors (p_{k-1}, p_k) and (p_k, p_{k+1}) is assumed to not exceed this threshold (for exceptions see [5]).

`Da_min` A minimal value for the product $\Delta_k \alpha$, where Δ_k is the current step size and α is the angle between (p_{k-1}, p_k) and (p_k, p_{k+1}) (see [5]).

`Da_max`	A maximal value for the product $\Delta_k\alpha$, where Δ_k is the current step size and α is the angle between (p_{k-1}, p_k) and (p_k, p_{k+1}) (see [5]).
`D_min`	A minimal step size.
`eB`	A search circle bisection error, *i. e.* if the distance between the image of $\hat{p}_{k+1}$ (where $\hat{p}_{k+1}$ is a candidate for the next manifold point p_{k+1}) and a certain segment (p_{i-1}, p_i) is less than `eB`, the point p_{k+1} is assumed to be found and the bisection stops.
`....`	The parameters of the iterated map, which may be specified in two ways. They can be given either as a simple comma separated list, or as a single cell-array parameter (see *example?*, note that `{:}` is important).

To understand more deeply the role of the parameters `a_min`, `a_max`, `Da_min`, `Da_max`, `D_min`, and `eB` one is encouraged to refer to [5] and other works of the same authors.

Return values. The function returns the following values:

`p`	An array of the computed manifold points.
`arc_length`	A computed manifold arc length

Displayed output. While running, the function displays some output in the Command Window. Information about the calculation progress is printed in blue, warnings (*e. g.*, about inability to find the next point, or acceptance the point not satisfying the required constraints—for details see [4]) appear in green. Certain auxiliary information is displayed in orange, and the final data is printed in violet. (For colorizing the output the utility `cprintf` is used [2]).

The example for usage of `sc_stable` is presented in Listing 19.

5.2.2 Periodic points

To compute the stable manifold of a periodic point with period greater than one, the function `sc_stable_cyc` should be used (`sc_stable_cyc.m`).

Listing 18: sc_unstable.m

```
function [p, arc_length] = sc_stable_cyc(fhandle, p0, ...
  period, arc_max, max_iter, d, a_min, a_max, Da_min, ...
  Da_max, D_min, eB, ....)
%
% Find a 1D stable manifold of a saddle point
....
```

Parameters. The function requires the following parameters:

`fhandle`	A function handle corresponding to the desired map (for more details on function handles see [1]).
`p0`	A saddle point whose manifold is to be computed.
`period`	A period of the target saddle point.
`arc_max`	A desired length of the manifold arc (is approximated as a sum of vector norms $\|p_{k+1} - p_k\|$, where $\{p_k\}_{k=0}^{N}$ are computed points of the manifold)
`max_iter`	A maximal number of iterates to try. This parameter is useful in case of a non-invertible map, which may have several preimages. The stable manifold of such a map may be mapped onto itself in a compex way, so that it is required to iterate the map several times until getting the right approximation for the next point (see [5, Example 4.3]).
`d`	An initial step size. Note that if `d` > 0 manifold is grown forward, while if `d` < 0 it is grown backward.
`a_min`	A minimal angle value. If the angle α between the vectors (p_{k-1}, p_k) and (p_k, p_{k+1}) falls below this value, the step size is increased (see [5]).
`a_max`	A maximal angle value. The angle α between the vectors (p_{k-1}, p_k) and (p_k, p_{k+1}) is assumed to not exceed this threshold (for exceptions see [5]).

`Da_min` A minimal value for the product $\Delta_k\alpha$, where Δ_k is the current step size and α is the angle between (p_{k-1}, p_k) and (p_k, p_{k+1}) (see [5]).

`Da_max` A maximal value for the product $\Delta_k\alpha$, where Δ_k is the current step size and α is the angle between (p_{k-1}, p_k) and (p_k, p_{k+1}) (see [5]).

`D_min` A minimal step size.

`eB` A search circle bisection error, *i. e.* if the distance between the image of $\hat{p}_{k+1}$ (where $\hat{p}_{k+1}$ is a candidate for the next manifold point p_{k+1}) and a certain segment (p_{i-1}, p_i) is less than `eB`, the point p_{k+1} is assumed to be found and the bisection stops.

`....` The parameters of the iterated map, which may be specified in two ways. They can be given either as a simple comma separated list, or as a single cell-array parameter (see *example?*, note that `{:}` is important).

Return values. The function returns the following values:

`p` An array of the computed manifold points.

`arc_length` A computed manifold arc length

Displayed output. While running, the function displays some output in the Command Window. Information about the calculation progress is printed in blue, warnings (*e. g.*, about inability to find the next point, or acceptance the point not satisfying the required constraints—for details see [4]) appear in green. Certain auxiliary information is displayed in orange, and the final data is printed in violet. (For colorizing the output the utility `cprintf` is used [2]).

The example for usage of `sc_stable` is presented in Listing 19.

Listing 19: do_stab_sample.m

```
% modified Ikeda map
param = {1, 0.9, 1, 0.4, 6};
p0 = [1.08331887404 -2.4079634834];
[sf1, arc] = sc_stable(@ikeda, p0, 10, 1, 0.001, 0.2, ...
             0.3, 1e-6, 1e-5, 1e-4, 1e-6, param{:});
[sb1, arc] = sc_stable(@ikeda, p0, 10, 1, -0.001, ...
             0.2, 0.3, 1e-6, 1e-5, 1e-4, 0.2, param{:});
....
% shallow map, period-2 point
%!!! another example, this does not work!
p0 = [2.13884578746 -0.46130133276];
[sf, arc] = sc_unstable_cyc(@shallow, p0, 2, 30, ...
     0.001, 0.2, 0.3, 1e-6, 1e-5, 1e-4, 0.2, param{:});
[sb, arc] = sc_unstable_cyc(@shallow, p0, 2, 30, ...
     -0.001, 0.2, 0.3, 1e-6, 1e-5, 1e-4, 0.2, param{:});
```

5.3 Plotting computed manifolds

To plot computed manifolds together with the periodic points found one should use the function `plot_manifolds` (found in `plot/plot_manifolds.m`).

Listing 20: plot_manifolds.m

```
function hfig = plot_manifolds(points, umanif, smanif)
%
% Plot previously calculated (un)stable manifolds
....
```

Parameters. The function requires the following parameters:

`points` The cell array of periodic points, which should be of the same secification as one obtained from the functions `find_periodic_1d` and `find_periodic_md`.

`umanif` The cell array of unstable manifolds to be plotted (may be empty `{}`).

`smanif` The cell array of stable manifolds to be plotted (may be empty `{}`).

Return values. The function returns the following values:

`hfig` A handle for the plotted figure window.

For the example of usage of `plot_manifolds` see Listing 21 (`examples/do_manif_sample.m`).

Listing 21: do_manif_sample.m

```
% modified Ikeda map
param = {1, 0.9, 1, 0.4, 6};
xF = find_periodic_md(@ikeda, 2, [-5 5; -5 5], [10 10],
    ...
          1e-8, 50, false, param{:});
points = period_classify(@ikeda, xF, 10, 1e-6, param{:});
fixed = points{2, 1}(1, :);
cyc1 = points{1, 1}(1, :);
cyc2 = points{1, 1}(2, :);
[u_fix_f, arc] = sc_unstable(@ikeda, fixed, 10, 0.001,
    ...
0.2, 0.3, 1e-6, 1e-5, 1e-4, 0.2, param{:});
....
[s_cyc2_b, arc] = sc_stable_cyc(@ikeda, cyc2, 2, 10, ...
1, -0.001, 0.2, 0.3, 1e-6, 1e-5, 1e-4, 1e-6, param{:});
% construct cell arrays of computed manifolds
umanif = {u_fix_f, u_fix_b, u_cyc1_f, u_cyc1_b, u_cyc2_f,
    u_cyc2_b};
smanif = {s_fix_f, s_fix_b, s_cyc1_f, s_cyc1_b, s_cyc2_f,
    s_cyc2_b};
% plot points and manifolds
hfig = plot_manifolds(points, umanif, smanif);
hfig = plot_manifolds({points{1, :}; points{2, :}}, ...
                      {}, smanif);
```

A File listing

File name	Short description
`init.m`	Initialization file (see Sec. 1)
`data`	Directory with Matlab data files for defining the colour palette and certain default parameter values
`bd_colors.mat`	Matlab data file defining the colour palette for plotting 2D period diagrams
`default.mat`	Matlab data file containing default method parameter values and several auxiliary variables
`doc`	Directory with documentation
`description.pdf`	Routine user guide
`example`	Directory with sample files
`do_bd_sample.m`	sample Matlab code for plotting 2D bifurcation
`do_manif_sample.m`	sample Matlab code for finding and plotting invariant manifolds (using Ikeda map)
`do_points_sample.m`	sample Matlab code for finding periodic points
`do_stab_sample.m`	sample Matlab code for growing stable manifolds
`do_unstab_sample.m`	sample Matlab code for growing unstable manifolds
`main`	Directory with main routine functions
`find_periodic_1d.m`	find periodic points of the given period in 1D maps
`find_periodic_md.m`	find periodic points of the given period in mD maps, with $m > 1$
`period_bd_calc.m`	calculate the data for a 2D period diagram

File name	Short description
period_classify.m	classify the periodic points found by find_periodic_1d.m or find_periodic_md.m, and arrange them in cycles
sc_stable_cyc.m	find a 1D stable manifold of a saddle periodic point, uses the search circle algorithm
sc_stable.m	find a 1D stable manifold of a saddle fixed point, uses the search circle algorithm
sc_unstable_cyc.m	find a 1D unstable manifold of a saddle periodic point, uses the search circle algorithm
sc_unstable.m	find a 1D unstable manifold of a saddle fixed point, uses the search circle algorithm
map	Directory with map definition functions
cubic.m	cubic map
cutting_map.m	cutting tool map (highly interrupted map)
gclm.m	globally coupled logistic maps, dimension is defined as a parameter
henon_J.m	Jacobi matrix for the Hénon map in the given point
henon.m	Hénon map
ikeda_J.m	Jacobi matrix for the modified Ikeda map
ikeda.m	modified Ikeda map
Lambda_noinv.m	a certain non-invertible map
logistic_J.m	derivative of the logistic map
logistic.m	logistic map
pwl3part.m	piecewise linear map defined on 3 partitions

File name	Short description
shallow.m	shallow map
skewtent.m	skew tent map
plot	Directory with plotting functions
bd_plot.m	plot colour-coded period diagram
plot_manifolds.m	plot periodic points and their invariant manifolds
util	Directory with auxiliary functions
bisection.m	bisection method for the search circle algorithm (used in calculating stable manifolds)
calc_jacob_cyc.m	calculate jacobian for a periodic point with period larger than one
check_segment.m	check if the given point lies inside the target segment
dist_point_segm.m	find distance between the given point and a target segment
find_angle.m	find angle between the given vector and the vertical axis
find_delta.m	find step size for the search circle algorithm
find_period.m	find period of the given point
find_segment.m	find the segment which contains the preimage of the next manifold point (used in the search circle algorithm)
get_cartesian.m	calculate the cartesian coordinates of the end point for the vector given by polar-like coordinates (the radius and the clockwise angle between the vector and the vertical axis)
get_colour.m	get appropriate colour for colorizing text output

File name	Short description
`is_acute.m`	check if the angle between the two specified vectors is acute ($< \pi$)
`is_clockwise.m`	check if the given point is situated clockwise with respect to the target vector
`iterate.m`	iterate the function given number of times
`jacob.m`	calculate approximately the Jacobian matrix in the given point
`make_subdiv.m`	divide a multi-dimensional domain into a number of subdomains
`search_circle.m`	the search circle algorithm for growing a 1D invariant manifold
`vector_angle.m`	find the angle between the two vectors
`util/cprintf`	cprintf utility source for producing colorized output (see [2])

B Search circle algorithm

The method used to construct invariant manifolds is the Search Circle Algorithm (SCA) [4, 5], which grows a one-dimensional manifolds in steps by adding new points according to the local curvature properties of the manifold. The difference of the SC algorithm from other methods is that it **does not need the inverse**. It finds a new point close to the last computed point that maps under the function f to a piece of the manifold that was already computed.

B.1 Stable manifold

B.1.1 Growing the manifold

The algorithm produces a piecewise linear approximation of $W^s(p_0)$ by computing successive points $M = \{p_0, p_1, \ldots, p_N\}$ at varying distance from each other. The first point p_1 is taken at a small distance $\delta > 0$ from p_0 along

the stable eigenspace $E^s(p_0)$. The distance between consecutive points is adjusted according to the curvature of the manifold.

Suppose that the manifold has been grown up to some point p_k such that $M = \{p_0, p_1, \ldots, p_k\}$. We wish to find the next point p_{k+1} at a distance Δ_k from p_k. The manifold is forward invariant, so new points, and in particular p_{k+1}, must map onto the piece of the manifold that has been computed so far. Thus, we are trying to find a point p_{k+1} at a distance Δ_k from p_k such that $f(p_{k+1})$ belongs to some segment of already computed part of the manifold[1]. For that we draw a circle $C(p_k, \Delta_k)$ of the radius Δ_k with a center in p_k (green colour in Fig. 1). Provided that Δ_k is small enough, the circle $C(p_k, \Delta_k)$ intersects the manifold $W^s(p_0)$ only twice, namely, in the point p^* (which belongs to the already computed segment, and thus, is not the target point) and the target point p_{k+1}. The image $f(C(p_k, \Delta_k))$ is then a closed curve (magenta colour in Fig. 1), which is located near some segment of the already computed manifold. Obviously, $f(C(p_k, \Delta_k))$ also has two intersections with $W^s(p_0)$, from which the one $f(p^*)$ we are not interested in (in Fig. 1 it belongs to (p_{i-2}, p_{i-1})), while the other $f(p_{k+1})$ (belonging to (p_i, p_{i+1})) is the image of the point p_{k+1} searched for.

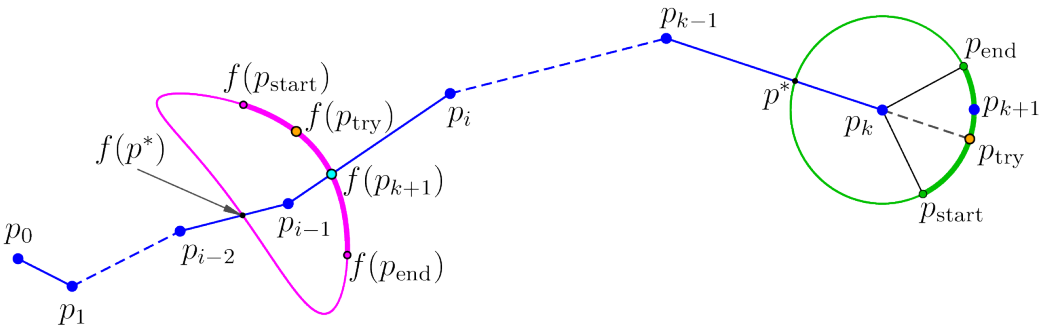

Figure 1: Manifold piecewise linear approximation by the SCA

The distance Δ_k is chosen so that it is acceptable meaning that the interpolation error is within the desired accuracy which ensures that the approximated manifold is of reasonable resolution. For that the method suggested in [6] is used, namely, it is assumed that the angle α_k between the lines drawn through p_{k-1}, p_k and p_k, p_{k+1} (see Fig. 2) lies whithin ceratin bounds

[1]Note, that in certain cases of non-invertible maps, the point p_k is mapped to the previously calculated part of the manifold not by f, but by some its iteration f^m.

$\alpha_{\min} < \alpha_k < \alpha_{\max}$. If $\alpha_k > \alpha_{\max}$, the point p_{k+1} is too far apart the point p_k, so the step should be reduced, and the procedure is repeated with the decreased step. If $\alpha_k < \alpha_{\min}$, the point p_{k+1} is too close to p_k, however, for making the decision of enlarging the step or not we check also another condition $\Delta_k \alpha_k > (\Delta\alpha)_{\min}$, which is explained below.

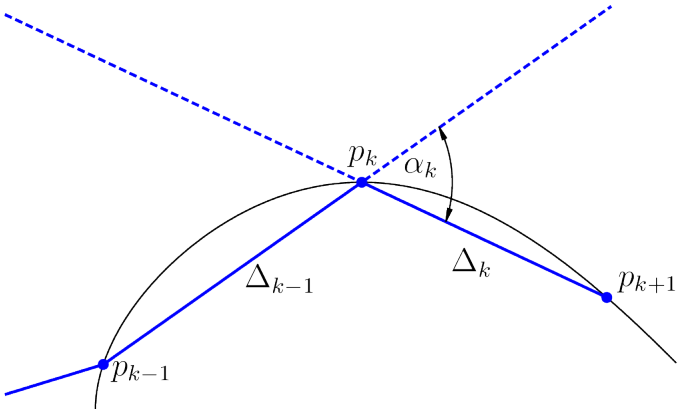

Figure 2: The angle between the lines drawn through p_{k-1}, p_k and p_k, p_{k+1}.

We can ensure immediately that α_k does not exceed $\alpha_{\max}$ by only searching the part of the circle $C(p_k, \Delta_k)$ that satisfies this criterion. This search region is the arc between the points p_{start} and p_{end}, indicated by thicker green curve in Fig. 1, and its image, which is the thicker magenta curve between $f(p_{\text{start}})$ and $f(p_{\text{end}})$, intersects the previously computed part of the manifold only once, which automatically ensures that we do not accidentally search for a pre-image of the point p^* (that we are not interested in).

In addition, to control the local interpolation error, we also check that the product $\Delta_k \alpha_k$ lies between $(\Delta\alpha)_{\min}$ and $(\Delta\alpha)_{\max}$. If $\Delta_k \alpha_k > (\Delta\alpha)_{\max}$, the step Δ_k is halved and the procedure of finding p_{k+1} is repeated. If $\Delta_k \alpha_k < (\Delta\alpha)_{\min}$ together with $\alpha < \alpha_{\min}$, then for the next iteration the step is enlarged $\Delta_{k+1} = 2\Delta_k$. This ensures that the number of points used to approximate the manifold is in some sense optimized for the required accuracy constraints. Note that, at sharp folds it may be necessary to accept $\alpha_k > \alpha_{\max}$ due to Δ_k becoming very small. In this case we accept the "inacceptable" point if $\Delta_k < \Delta_{\min}$, where $\Delta_{\min}$ is also a predefined parameter.

Finally, in order to find p_{k+1} we need to define which segment of the previously calculated manifold contains the intersection point with $f(C(p_k, \Delta_k))$. We first try the segment (p_{i-1}, p_i) that was used in the previous step (to find

a candidate for p_k). If the image $f(p_{k+1})$ of the candidate for p_{k+1} lies on the line through p_{i-1} and p_i, but not in the segment (p_{i-1}, p_i), we discard this point and repeat the algorithm with the following segment (p_i, p_{i+1}) to find a new candidate for p_{k+1}. (If the map has multiple pre-images then we may need to search for $f(p_{k+1})$ on the previous segment (p_{i-2}, p_{i-1})). To find the candidate for p_{k+1} we use the bisection method for the angle α_k, and allow the point $f(p_{k+1})$ to lie at a maximum distance of some small ε_B (the bisection error) from the detected segment, say, $I_j = (p_j, p_{j+1})$, of the previously computed manifold part. Note, that in this procedure the points $f(p_{\text{start}})$ and $f(p_{\text{end}})$ must lie on the opposite sides of the segment I_j. If they do not (for instance, if there is a sharp fold in the manifold) the search region for the angle α_k is increased, and a warning message is printed.

To sum up, a single run of the algorithm may be briefly described by the following steps:

1. We put $\Delta_k = \Delta_{k-1}$ and draw a circle $C(p_k, \Delta_k)$.

2. We take the image of the arc between p_{start} and p_{end} and check if $f(p_{\text{start}})$ and $f(p_{\text{end}})$ lie on the opposite sides of the line drawn through p_{i-1} and p_i. If not, we increase the search region for α_k.

3. We use bisection method for finding the candidate p_{k+1} such that $f(p_{k+1})$ lies whithin the distance ε_B from the line through p_{i-1}, p_i.

4. If $f(p_{k+1})$ lies outside the segment (p_{i-1}, p_i), we discard the point p_{k+1} and restart the procedure with using the segment (p_i, p_{i+1}).

5. If $f(p_{k+1})$ lies inside the segment (p_{i-1}, p_i), we check the condition $\Delta_k \alpha_k < (\Delta\alpha)_{\max}$, and if it fails, we assign $\Delta_k = \Delta_k/2$ and restart the procedure.

6. If $\Delta_k \alpha_k < (\Delta\alpha)_{\max}$, then we check also that $\Delta_k \alpha_k > (\Delta\alpha)_{\min}$. If not and additionally $\alpha_k < \alpha_{\min}$, we accept the found point, but assign $\Delta_{k+1} = 2\Delta_k$ for the next run of the algorithm.

7. Finally, if the angle search region was enlarged and $\alpha_k > \alpha_{\max}$, we also check if $\Delta_k < \Delta_{\min}$. If yes, we accept the found point ignoring the failed criterion.

B.2 Unstable manifold

the procedure for the unstable manifold is quite similar, with the main difference that now mapping is in the reverse direction. So that, having $M = \{p_0, p_1, \ldots, p_k\}$ already computed, we would like to find a point p_{k+1} such that some point q_{k+1}, which belongs to a certain already known segment $I_i = (p_{i-1}, p_i)$, is mapped to p_{k+1} ($f(q_{k+1}) = p_{k+1}$). We also assume that the point p_{k+1} lies at an approximate distance Δ_k from p_k (see Fig. 3).

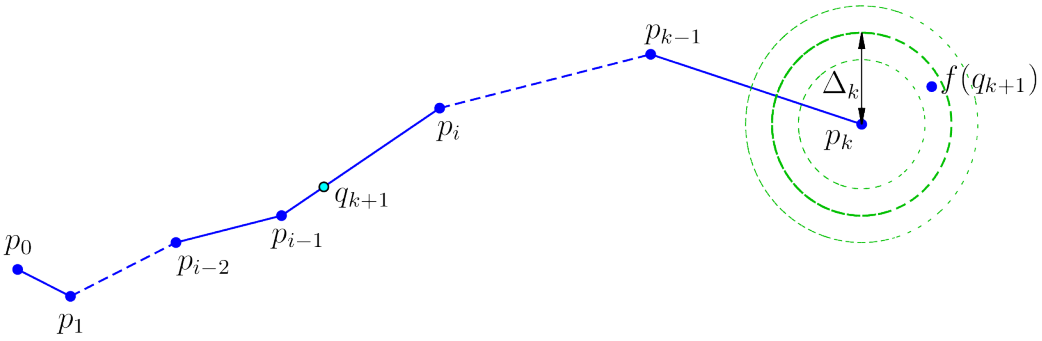

Figure 3: Unstable manifold approximation

Then, to find the candidate for p_{k+1}, we find the images of the points of the segment I_i, which are mapped to some neighbourhood of p_k. We use the bisection to detect the point q_{k+1} whose image is localted near the point p_k so that $\Delta_k - \varepsilon_B < \|f(q_{k+1}) - p_k\| < \Delta_k + \varepsilon_B$. If such a point does not exist, we take the next segment $I_{i+1} = (p_i, p_{i+1})$ and repeat the procedure.

Finally, we check the same accuracy conditions

$$\alpha_{\min} < \alpha_k < \alpha_{\max}, \quad (\Delta\alpha)_{\min} < \Delta_k \alpha_k < (\Delta\alpha)_{\max},$$

adjusting the step as described for the stable manifold case, and also accept the point p_{k+1} if $\alpha_k > \alpha_{\max}$, but $\Delta_k < \Delta_{\min}$.

References

[1] MathWorks documentation center, Matlab online help, http://www.mathworks.com/help/matlab.

[2] cprintf utility, downloadable from http://www.mathworks.com/matlabcentral/fileexchange/24093

[3] *Numerical Recepies in C*, Second Edition (1992).

[4] B. Krauskopf, H. M. Osinga, Growing 1D and quasi 2D unstable manifolds of maps, *J. Comput. Phys.* **146**(1), pp. 404–419 (1998).

[5] J. P. England, B. Krauskopf, H. M. Osinga, Computing one-dimensional stable manifolds of planar maps without the inverse, *SIAM Journal on Applied Dynamical Systems* **3**(2), pp. 161–190 (2004).

[6] D. Hobson, An efficient method for computing invariant manifolds of planar maps, *J. Comput. Phys.* **104**, pp. 14–22 (1993).