PRACTICUM II

SUMMER SCHOOL UTRECHT:
APPLIED BIFURCATION THEORY

Wednesday 17 July 2019

MAIKEL M. BOSSCHAERT

Contents

1 DDE-BifTool demo: time-delayed systems with band-limited feedback

DDE-BifTool[1] is a set of routines for performing numerical bifurcation analysis of delay differential equations running in MATLAB or GNU Octave[2]. One can download dde_biftool_v3.2a from the DDE-BifTool website and extract its contents. A minimal version, without the demos, can be downloaded from the Utrecht Summer School 2019 webpage. By listing the contents of the folder, we obtain the following sub-folders:

ddebiftool Basic DDE-BifTool routines.

ddebiftool_extra_nmfm Extension for normal form computations at local bifurcations of equilibria in DDEs with constant delay. Formally, also discrete state-dependent DDEs are supported.

ddebiftool_extra_psol Extension for local bifurcations of periodic orbits.

ddebiftool_extra_rotsym Extension for systems with rotational symmetry.

ddebiftool_extra_symbolic Extension generating a system file and higher order derivatives of the system.

ddebiftool_utilities Various auxiliary functions.

demos Example scripts illustrating the use of DDE-BifTool (a new folder under this directory is an appropriate location for adding additional examples)

doc Manual for DDE-BifTool.

external_tools (Deprecated) Mathematica and a Maple script for generating derivative functions used in DDE-BifTool.

1.1 Tutorial overview

In this short tutorial, we will show how to:

- Generate the system file containing the system to be analyzed as well as the higher order derivatives needed for the normal form computations.
- Setup a steady-state and compute its stability.
- Continue a branch of equilibria.
- Detect a Hopf bifurcation along the branch.
- Compute the first Lyapunov coefficient of a Hopf point.

[1] https://sourceforge.net/projects/ddebiftool/
[2] http://www.gnu.org/software/octave

- Continue a branch of periodic solutions emanating from a detected Hopf point.
- Determine the stability of the periodic solutions.
- Detect period doubling bifurcations.
- Continue period doubling bifurcation in two parameters.
- Continue Hopf point in two parameters.
- Create a two-parameter bifurcation diagram.
- Plot the time profiles of periodic solutions.

We will use the following congestion control model, treated in [2] and [1],

$$\begin{cases} \dot{w}(t) = 1 - \dfrac{w(t)w(t-1)}{2} kq(t-1), \\ \dot{q}(t) = w(t) - c. \end{cases} \tag{1}$$

Here k and c are parameters.

1.2 The system file

Before we start using `DDE-BifTool`, we first need to create the *system file*.

To create the system file:

(i) Create a new directory `congestionControlModel` in the demo folder of `DDE-BifTool`.

(ii) Enter the newly created directory and create a file `gen_sym_congestionControlModel.m`.

(iii) First read Listing 1 and then copy the text into the file `gen_sym_congestionControlModel.m`.

(iv) Execute the script `gen_sym_congestionControlModel` to generate the system file.

(v) Lastly, verify that the file `sym_congestionControlModel` has been created.

Listing 1: `gen_sym_congestionControlModel`

```
%% Congestion control model
% generate right-hand side and derivatives from symbolic expressions
%
% From: Hollot, C. V. and Chait, Y.
% Nonlinear stability analysis for a class of TCP/AQM networks
% Proceedings of the 40th IEEE Conference on Decision and Control,
% 3:2309-2314, 2001
%
%% Differential equations
%
% $$\dot w(t)=1-\frac{w(t) w(t-1)}{2} k q(t-1),$$
```

```
%
% $$\dot q(t)=w(t)-c.$$
%
% Parameters are (in this order) |k|, |c|, |tau|
```

```
%% Add paths and load sym package if GNU Octave is used
clear
ddebiftoolpath='../../';
addpath(strcat(ddebiftoolpath,'ddebiftool'),...
    strcat(ddebiftoolpath,'ddebiftool_extra_symbolic'));
if dde_isoctave()
    pkg load symbolic
end
```

```
%% Create parameter names as strings and define fixed parameters
% The demo has the parameters |k|, |c| and |tau|
parnames={'k','c','tau'};
```

```
%% Create symbols for parameters, states and delays states
% The array |par| is the array of symbols in the same order as parnames.
% Due to the following two lines we may, for example, use either k or
% par(1) to refer to the delay.
syms(parnames{:});       % create symbols for k, c and tua
par=cell2sym(parnames);  % now gamma is par(1) etc
```

```
%% Define system using symbolic algebra
syms w wtau q qtau % create symbols for w(t) w(t-tau), q(t), q(t-tau)
dw_dt=1-w*wtau/2*k*qtau;
dq_dt=w-c;
```

```
%% Differentiate and generate code
[fstr,derivs]=dde_sym2funcs(...
[dw_dt;dq_dt],... % n x 1 array of derivative symbolic expressions
[w,wtau;q,qtau],... % n x (ntau+1) array of symbols for states
par,... % 1 x np (or np x 1) array of symbols used for parameters
'filename','sym_congestionControlModel');  % argument output file
```

1.3 Initialize the system

Now that the system file is created, we start using `DDE-BifTool` to analyze system (1) numerically. For this, create a new file `congestionControlModel.m` in the directory `congestionControlModel`. The code in the listings below should be copied in this file. Then execute the code and compare with the results shown here.

Listing 2: Initialize the system

```
%% Utrecht Summer School 2019 ABT Demo Congestion control model
%
% Demo illustrating how to branch off a Hopf point, branch off period
% doubling bifurcations and continue Hopf bifurcation curve in two
% parameters. Furthermore, the stability of the various branches
% is computed.
%
```

```
%% Differential equations
%
% From: Hollot, C. V. and Chait, Y.
% Nonlinear stability analysis for a class of TCP/AQM networks
% Proceedings of the 40th IEEE Conference on Decision and Control,
% 3:2309-2314, 2001
%
% $$\dot w(t)=1-\frac{w(t) w(t-1)}{2} k q(t-1),$$
%
% $$\dot q(t)=w(t)-c.$$
%
% Parameters are (in this order) |k|, |c|, |tau|
```

```
%% load DDE-BifTool into MATLAB path
clear
close all
ddebiftoolpath='../../';
addpath(strcat(ddebiftoolpath,'ddebiftool'),...
    strcat(ddebiftoolpath,'ddebiftool_extra_psol'),...
    strcat(ddebiftoolpath,'ddebiftool_extra_nmfm'),...
    strcat(ddebiftoolpath,'ddebiftool_utilities'));
format compact
set(groot, 'defaultTextInterpreter', 'LaTeX');
```

```
%% Initial parameters and state
parnames={'k','c','tau'};
cind=[parnames;num2cell(1:length(parnames))];
ind=struct(cind{:});
bounds={'max_bound',[ind.k 8;ind.c 2.5],'max_step',[0,0.3],...
    'min_bound',[ind.k 2]};
```

```
%% Set user-defined functions
%  use the right-hand side and derivatives created via symbolic toolbox
funcs=set_symfuncs(@sym_congestionControlModel,'sys_tau',@()ind.tau);
```

The `funcs` structure contains the system definition and higher order derivatives previously generated by the script `gen_sym_congestionControlModel`. This structure is passed along the various routines of `DDE-BifTool`.

1.4 Setup a steady-state and compute its stability

As an initial steady-state, we take the solution $(w,q) \equiv (c, \frac{2}{kc^2})$ with parameter values $k = 2, c = 1$ and $\tau = 1$. In `DDE-BifTool`, a steady-state is represented by a structure, which should be created with the function `dde_stst_create`. To compute the stability of the steady-state, we use the function `p_stabil`. The arguments are the `funcs` structure, the steady-state, and a structure containing various parameters used in the computation of the stability. This last structure is created with the function `df_mthod`. As seen below, we adjust the parameter `method_stst.stability.minimal_real_part` to -3. Then all eigenvalues, which have minimal real part greater than or equal to -3, are computed. Determine the default value of `method_stst.stability.minimal_real_part`. Lastly, we

plot the eigenvalues, see Figure 1.

```
%% Construct steady-state point
k=2; c=1;
stst=dde_stst_create('x',[c;2/(k*c^2)]);
stst.parameter(ind.k)   = k;
stst.parameter(ind.c)   = c;
stst.parameter(ind.tau) = 1;
```

```
% Compute stability
method_stst=df_mthod(funcs,'stst');
method_stst.stability.minimal_real_part=-3;
stst.stability=p_stabil(funcs,stst,method_stst.stability);
```

```
% Plot eigenvalues
figure(1); clf;
plot(stst.stability.l1,'*')
title('Stability plot of stst')
xlabel('$\Re(\lambda)$')
ylabel('$\Im(\lambda)$')
```

Alternatively, we could inspect the `stst` structure to obtain the computed eigenvalues of the steady-state.

```
>> stst.stability.l1

ans =
 -0.318131505204749 + 1.337235701430625i
 -0.318131505204749 - 1.337235701430625i
 -1.000000000000137 + 0.000000000000000i
 -2.062277729598343 + 7.588631178472476i
 -2.062277729598343 - 7.588631178472476i
 -2.653191974038767 +13.949208334533285i
 -2.653191974038767 -13.949208334533285i
>>
```

We conclude that, since the leading eigenvalues are in the left half complex plane, the steady-state is stable.

1.5 Continue a branch of equilibria

In `DDE-BifTool`, a branch is represented by a structure containing three fields:

- `point`: Contains an array of points that are on the branch.
- `method`: Parameters are set for continuation, solving, and stability.
- `parameter`: Information about which parameters to be continued, the maximum step size, and boundaries for these parameters.

We initialize a new steady-state branch with the function `SetupStst`. The steady-state `stst` will be used as a first point. A second point will automatically be computed

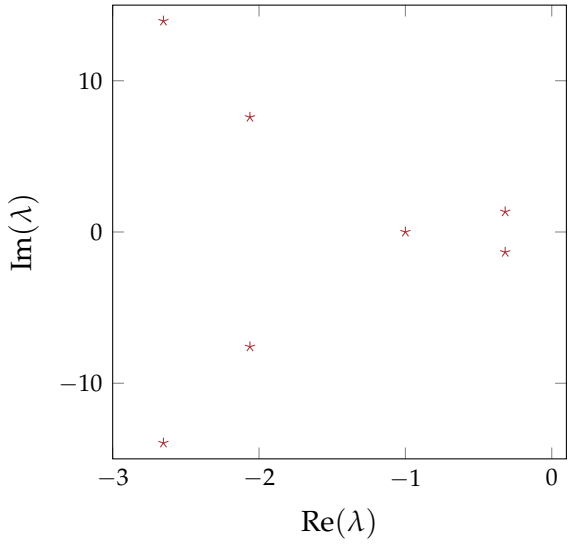


Figure 1: Stability plot of the steady-state `stst`. Real parts computed up to $\mathrm{Re}(\lambda) \geq -3$.

and added to the branch. Next, we continue the branch with the function `br_cont` for `n_steps=40` steps, or till when a boundary is encountered.

Listing 3: Branch of equilibria

```
%% Initialization of branch of steady-states
contpar=ind.k;
steadystate_br=SetupStst(funcs,'x',stst.x,'parameter',stst.parameter,...
    'step',0.1,'contpar',contpar,'max_step',[contpar,0.3],bounds{:});
%% Continue steady-state branch
figure(2);clf;ax2=gca;
n_steps=40;
steadystate_br=br_contn(funcs,steadystate_br,n_steps,'plotaxis',ax2);
xlabel('$k$')
ylabel('$c$')
```

The parameters of the steady-state solutions will be plotted during continuation, see Figure 2.

1.6 Detect a Hopf bifurcation along the branch

To determine if any bifurcation are present on the steady-state branch, we have to inspect the eigenvalues and see if any cross the imaginary axis. The function `LocateSpecialPoints` automatically computes the stability of the points on the branch and detects if the number of eigenvalues in the right half complex plane change. When the difference is equal to one or two, the routine will try to locate a fold, respectively a Hopf bifurcation. After successfully located a codimension one bifurcation point, its normal form coefficients are automatically computed, and the newly found point is added to the branch.

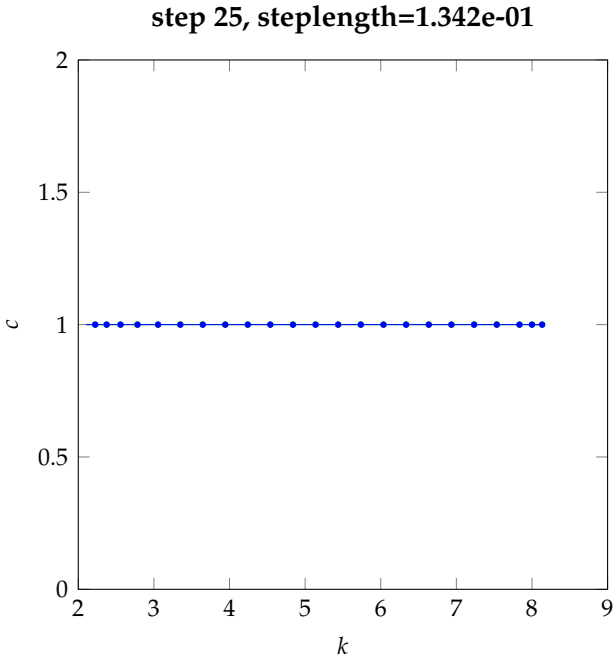


Figure 2: Branch of continued steady-state solutions of (1) keeping the parameter $c = 1$ fixed.

The function `GetStability` then calculates the number of unstable eigenvalues (or Floquet multipliers in the case of a branch with periodic solutions).

Listing 4: Detect bifurcations on steady-state branch

```
%% Detect bifurcations on steady-state branch
[steadystate_br,~,ind_hopf,bif1types]=LocateSpecialPoints(funcs,...
    steadystate_br);
nunst_eqs=GetStability(steadystate_br);
fprintf('Hopf bifurcation near point %d\n',ind_hopf);
```

In the MATLAB console, we see that a Hopf bifurcation is detected.

```
HopfCodimension2: (provisional)  detected.
StstCodimension1: calculate stability if not yet present
StstCodimension1: (provisional) 1 Hopf  detected.
br_insert: detected 1 of 1: hopf. Normalform:
    L1: -0.587787139308424
Hopf bifurcation near point 10
>>
```

Since the first Lyapunov coefficient `L1` is negative, the Hopf bifurcation is *supercritical*, and a stable periodic solution is predicted to emanate from the Hopf bifurcation point.

1.7 Compute the first Lyapunov coefficient of a Hopf bifurcation point

When a Hopf bifurcation is derived analytically, and one would like to confirm if the Hopf bifurcation is sub- or supercritical, then it is preferred to construct the Hopf bifurcation point directly from the steady-state `stst`. Since we haven't derived the conditions for a Hopf bifurcation, we inspect the located Hopf point on the steady-state branch `steadystate_br`.

```
>> steadystate_br.point(ind_hopf)
ans =
  struct with fields:

         kind: 'stst'
    parameter: [3.414105951101845 1 1]
            x: [2x1 double]
    stability: [1x1 struct]
         nmfm: [1x1 struct]
         nvec: [1x1 struct]
         flag: 'hopf'
>>
```

We adjust the parameter value and the position of the steady-state accordingly. Then we recompute the stability and convert the steady-state to a Hopf bifurcation point. Lastly, we compute its normal form coefficients with the function `nmfm_hopf`.

```
%% Construct Hopf point manually
stst.parameter(ind.k) = 3.414105951101845;
stst.x = [1;0.585804901384075];
% Recompute stability
method_stst=df_mthod(funcs,'stst');
stst.stability=p_stabil(funcs,stst,method_stst.stability);
% Convert steady-state to Hopf point and compute the normal form
hopf=p_tohopf(funcs,stst);
hopf=nmfm_hopf(funcs,hopf);
hopf.nmfm
```

The `MATLAB` console confirms again that the Hopf bifurcation is supercritical.

```
ans =
  struct with fields:

    L1: -0.587787139308631
>>
```

1.8 Continue the periodic solution emanating from a detected Hopf point

To continue the periodic solution emanating from the detected Hopf point along the steady-state branch `steadystate_br`, we use the function `SetupPsol` to initialize a branch with two points. Then we use the function `br_contn` to continue the periodic solutions, see Figure 3.

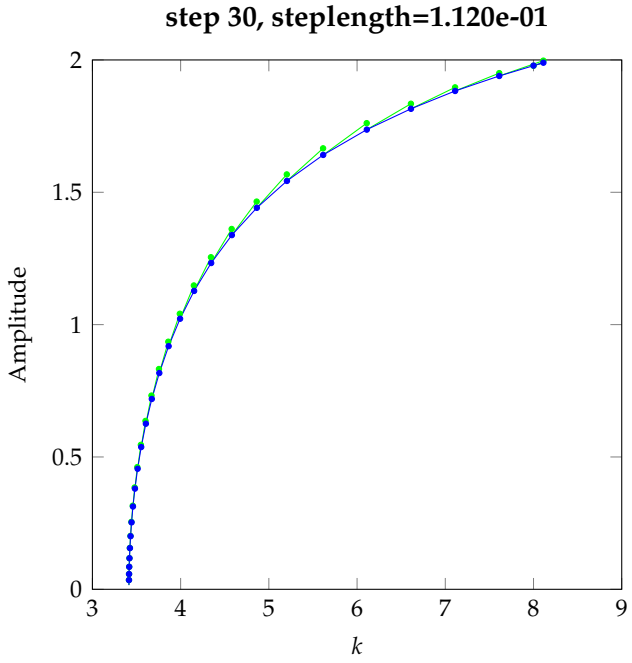


Figure 3: Continuation of periodic solutions emanating from the Hopf point detected on the steady-state branch.

Listing 5: Branch off at Hopf bifurcation

```
%% Branch off at Hopf bifurcation
disp('Branch off at Hopf bifurcation');
fprintf('Initial correction of periodic orbits at Hopf:\n');
[per1,suc]=SetupPsol(funcs,steadystate_br,ind_hopf,...
    'print_residual_info',1,'intervals',20,'degree',4,...
    'max_bound',[contpar,8],'max_step',[contpar,0.5],'matrix','full');
figure(3);clf;ax3=gca;
xlabel('$k$')
ylabel('Amplitude')
per1=br_contn(funcs,per1,60,'plotaxis',ax3);
```

1.9 Manual construction of periodic solution near Hopf bifurcation point

If it would be the case that we have constructed the Hopf bifurcation point manually, then we also have to construct a new periodic solution branch manually. First, we need to obtain a periodic solution near the Hopf bifurcation point. For this, we convert the Hopf point to a periodic solution with small amplitude using the function `p_topsol` and subsequently corrected the solution with the function `p_correc`. Note that, for correcting the periodic solution, we need to include a step condition `stpcond`, which ensures us that we do not converge back to the Hopf bifurcation point. Having corrected the periodic

solution, we compute its stability (multipliers) with the function `p_stabil`, similar as was done for the steady-state solution.

Listing 6: Manual construction periodic solution near Hopf point

```
%% Manually construct periodic solution near Hopf point
radius=0.1;
degree=4;
int_nr=20;
[psol,stpcond]=p_topsol(funcs,hopf,radius,degree,int_nr);
method=df_mthod('psol');
[psol,succ]=p_correc(funcs,psol,ind.k,stpcond,method.point)
```

```
%% Compute stability of periodic solution
psol.stability=p_stabil(funcs,psol,method.stability);
```

We inspect the `psol` structure to see the computed multipliers.

```
>> psol.stability.mu(1:6)
ans =
  1.000000000000022 + 0.000000000000000i
  0.981414537447360 + 0.000000000000000i
  0.000073971446841 + 0.000000000000000i
 -0.000005657390969 + 0.000049654020932i
 -0.000005657390969 - 0.000049654020932i
 -0.000001906656406 + 0.000002134469526i
>>
```

Since there is one trivial multiplier at 1 always present, we conclude that the periodic solution is stable, as predicted by the sign of the first Lyapunov coefficient. Using the function `p_splot`, with the perdiodic solution `psol` as its argument, we obtain Figure 4.

To continue the periodic solution, we create an empty periodic solution branch with the function `df_brnch`. The second argument is the parameter to be continued. Then two points (periodic solutions) need to be added. The first point will be an uncorrected periodic solution at the Hopf bifurcation point. The second will be the corrected periodic solution `psol`. We then continue the periodic solutions with the function `br_contn`. We obtain a nearly identical plot as in Figure 3.

```
%% Manual construction and continuation of psol branch
per_orb=df_brnch(funcs,ind.k,'psol'); % empty branch:
per_orb.parameter.max_bound=[ind.k 8];
per_orb.parameter.max_step=[ind.k 0.3];
deg_psol=p_topsol(funcs,hopf,0,degree,int_nr);
per_orb.point=deg_psol;
per_orb.point(2)=psol;
% compute periodic solutions branch
per_orb=br_contn(funcs,per_orb,60,'plotaxis',ax3);
xlabel('$k$');
ylabel('amplitude');
```

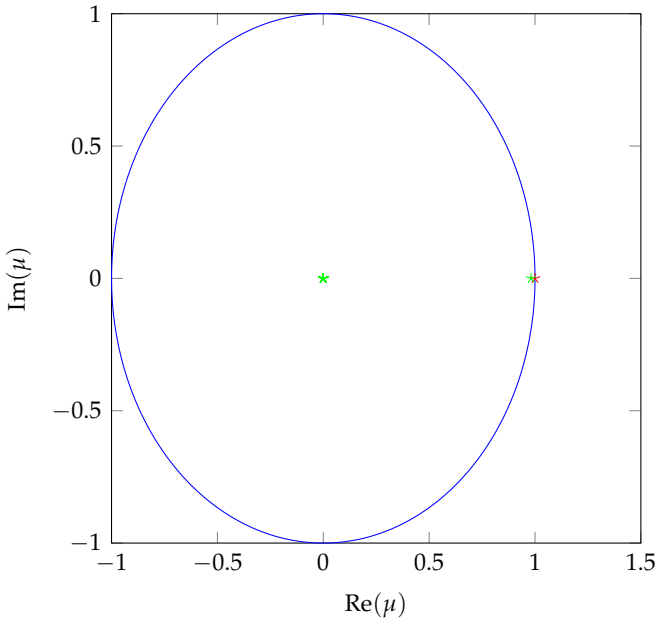


Figure 4: Multipliers of the periodic solution `psol` near the Hopf bifurcation `hopf`.

1.10 Detect period doubling bifurcation

Having obtained a branch of periodic solutions, either using the function `SetupPsol` or manually constructed, we now compute the stability along the branch with the function `br_stable`. After the stability has been computed, the function `GetStability` determines the number of unstable multipliers. For period doubling bifurcations, we search for differences in multipliers between the previous and the next point equal to 1, as shown in the last line in the listing below. For the Neimark-Sacker bifurcation, the difference should be equal to 2.

Listing 7: Detect period doubling bifurcation

```
%% Find period doubling bifurcation
per1=br_stabl(funcs,per1,0,1);
nunst_per=GetStability(per1,'exclude_trivial',true);
ind_pd=find(diff(nunst_per)==1);
```

By inspecting `ind_pd`, we see that there may be a period doubling bifurcation on the periodic solution branch `per1`.

1.11 Switching at a period doubling bifurcation

To switch to the period doubling branch, the function `DoublePsol` initializes a new branch with two corrected points of double period compared with the periodic solution `per1.point(ind_pd)` near the period doubling bifurcation. The function `br_contn` is used to continue the branch. By repeating this procedure two more times, i.e., continuing,

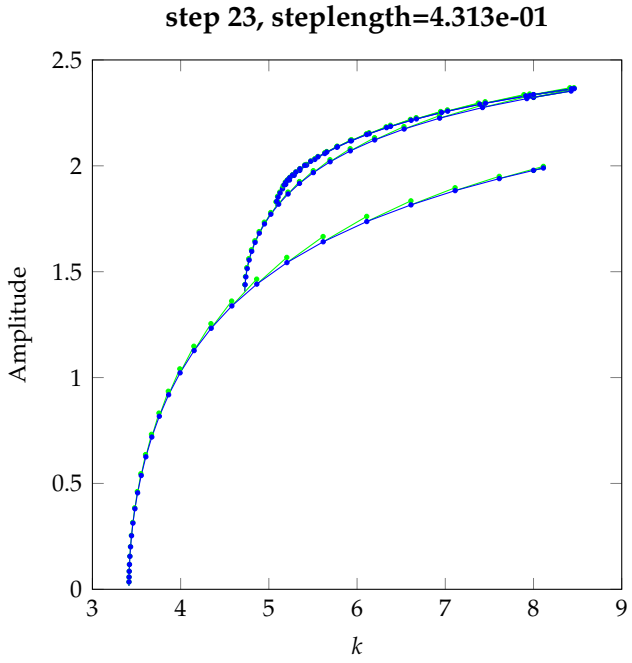


Figure 5: Multiple branches originating from the Hopf bifurcation point obtained by consecutively switching at period doubling bifurcation points. Note that there are four distinct branches, the last two are almost entirely overlapping.

computing stability, and searching for period doubling bifurcation, try to reproduce Figure 5.

```
%% Branch switching at period doubling bifurcation
ind_pd=find(diff(nunst_per)==1);
[per2,suc]=DoublePsol(funcs,per1,ind_pd(1));
per2=br_contn(funcs,per2,60,'plotaxis',ax3);
nunst_per2=GetStability(per2,'funcs',funcs,'exclude_trivial',true);
```

1.12 Continue period doubling bifurcation in two parameters

Instead of switching to the period doubling branch at a period doubling bifurcation point, we could also continue the period doubling bifurcation point in the two parameters (k, c). This is easily done by using the function `SetupPeriodDoubling`. Then, as usual, we continue the branch with `br_contn`. Try to reproduce Figure 6 by repeating this procedure for the other two detected period doubling bifurcation points.

```
%% Continue period doubling bifurcations in two parameters
[pdfuncs,pdbranch1,suc]=SetupPeriodDoubling(funcs,per1,ind_pd(1),...
    'contpar',[ind.k,ind.c],'dir',ind.k,'step',1e-1,bounds{:});
pdbranch1=br_contn(pdfuncs,pdbranch1,60,'plotaxis',ax2);
pdbranch1=br_rvers(pdbranch1);
```

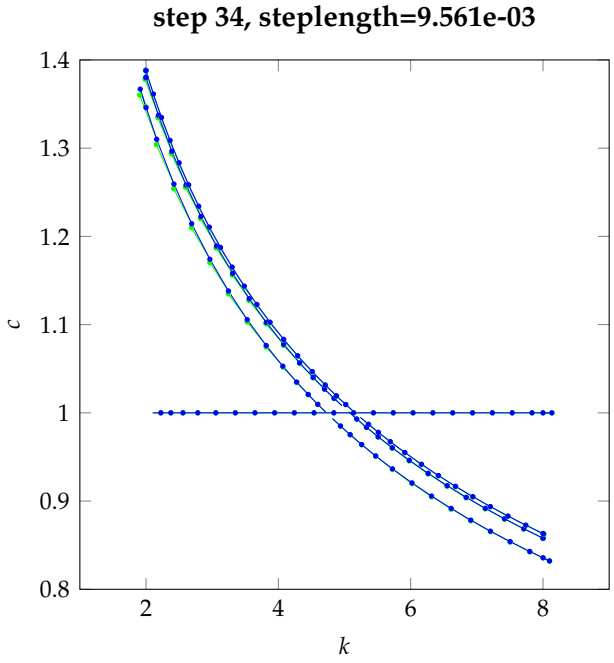


Figure 6: Continuation of multiple period doubling branches in the parameters (k, c). These branches emanate from the period doubling bifurcation points that have been detected.

```
pdbranch1=br_contn(pdfuncs,pdbranch1,60,'plotaxis',ax2);
nunst_pd1=GetStability(pdbranch1,'exclude_trivial',true,'funcs',pdfuncs);
xlabel('$k$'); ylabel('$c$')
```

1.13 Continue Hopf point in two parameters

In an almost identical manner, as in continuing the detected period doubling bifurcation in two parameters, we can also continue the Hopf bifurcation point in two parameters, as shown in the listing below. Then we use the function `LocateSpecialPoints` to determine the type of Hopf bifurcation points (sub- or supercritical) on the Hopf branch.

```
%% Continue Hopf bifurcation in two parameters
[hbranch,suc]=SetupHopf(funcs,steadystate_br,ind_hopf,...
    'contpar',[ind.k,ind.tau],'dir',ind.k,'step',1e-1,bounds{:});
hbranch=br_contn(funcs,hbranch,60,'plotaxis',ax2);
hbranch=br_rvers(hbranch);
hbranch=br_contn(funcs,hbranch,60,'plotaxis',ax2);
```

```
%% Compute L1 coefficient
% to find if Hopf bifurcation is supercritical (L1<0) or subcritical (L1>0)
[hbranch,hopftests,hc2_indices,hc2_types]=...
            LocateSpecialPoints(funcs,hbranch);
```

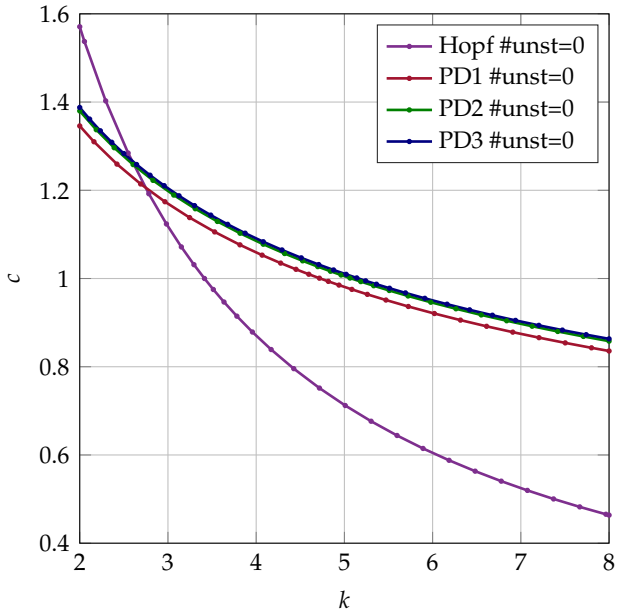


Figure 7: Bifurcation diagram in the parameters (k, c) of the congestion control model (1).

1.14 Bifurcation diagram in (k, c)

Using the function `Plot2dBranch`, a two-parameter bifurcation diagram is automatically constructed of the codimension one bifurcation curves, see Figure 7. Note that it is assumed that the stability along the various branches is computed.

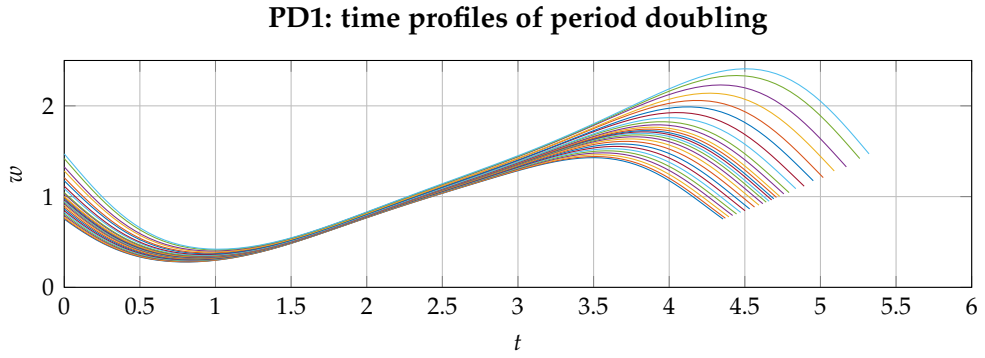
```
%% Two-parameter bifurcation diagram
% Assigning a name and color to a curve. Others are chosen automatically
figure(4); clf;ax3=gca;hold(ax3,'on')
lg=Plot2dBranch(hbranch);
lg=Plot2dBranch(pdbranch1,'lgname','PD1',...
    'funcs',pdfuncs,'oldlegend',lg);
lg=Plot2dBranch(pdbranch2,'lgname','PD2',...
    'funcs',pdfuncs,'oldlegend',lg,'color',[0,0.5,0]);
lg=Plot2dBranch(pdbranch3,'lgname','PD3',...
    'funcs',pdfuncs,'oldlegend',lg,'color',[0,0,0.5]);
title('Bifurcation diagram of congestion control model')
xlabel('$k$')
ylabel('$c$');
grid on
```

1.15 Compare periodic solutions on the period doubling branches

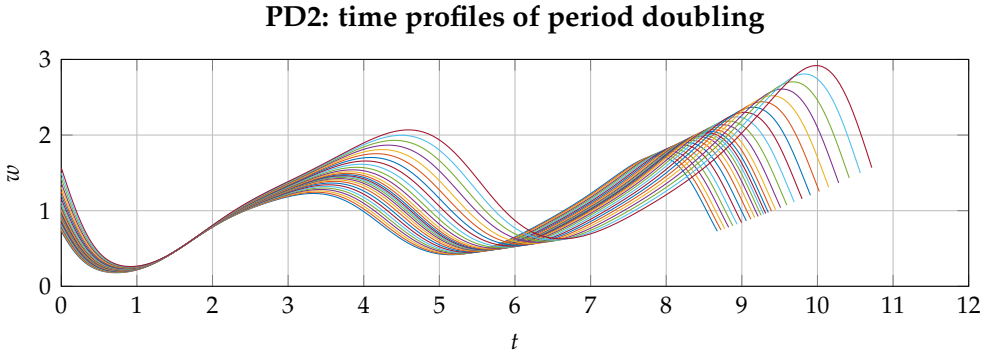
Lastly, we compare the periodic solutions on the continued period doubling branches by plotting the time profiles of $w(t)$.

```
%% Time profiles of the first component of the period doubling solutions
bifsols={pdbranch1.point,pdbranch2.point,pdbranch3.point};
figure(5); clf;
for k=1:3
    subplot(3,1,k);
    hold on
    for i=1:length(bifsols{k})
        plot(bifsols{k}(i).mesh*bifsols{k}(i).period,...
            bifsols{k}(i).profile(1,:),'-');
    end
    hold off
    box on
    grid on
    title(sprintf('PD%d: time profiles of period doubling',k));
    xlabel('$t$');
    ylabel('$w$');
 end
```

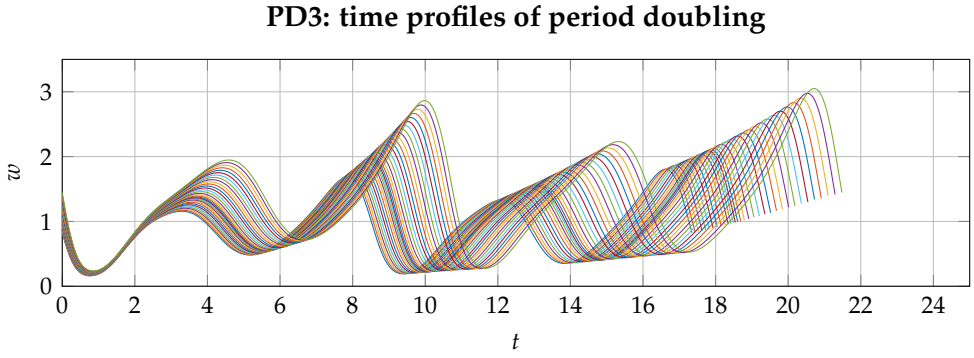
Figure 8 is clear numerical evidence of period doubling bifurcations.



(a)



(b)



(c)

Figure 8: Comparing the time profiles of $w(t)$ on the periodic doubling branches, the period doubling bifurcations are clearly visible.

2 Exercise

Consider again the delay differential equation

$$\dot{x}(t) = \left(-\frac{\pi}{2} + \mu\right) x(t-1)\,[1 + x(t)]\,, \tag{2}$$

treated in Practicum I.

2.1 First Lyapunov coefficient analytically

We will derive the first Lyapunov coefficient analytically. For notation used here, see today's slides. Below, all expressions are evaluated at the equilibrium $x(t) \equiv 0$, critical parameter value $\mu = 0$ and eigenvalue $\lambda = i\omega_0 = i\frac{\pi}{2}$.

(i) Calculate the (1×1 dimensional) characteristic matrix.

(ii) Using the characteristic matrix, calculate the *normalized* eigenfunctions. Note that the eigenvector and adjoint eigenvector of the characteristic matrix are not unique. Choose $q = 1$ and scale the eigenvector p.

(iii) Derive the (multi-)linear forms $D^2 f^0(Q, P)$ and $D^3 f^0(Q, P, R)$.

(iv) Calculate the quadratic coefficients h_{20} and h_{11}.

(v) Provide the normal form coefficient c_1.

(vi) Conclude from the first Lyapunov coefficient, that a stable periodic solution arises at the critical point.

2.2 First Lyapunov coefficient with DDE-BifTool

In this exercise, we will compute the first Lyapunov coefficient with DDE-BifTool.

(i) Open MATLAB.

(ii) In MATLAB go to the directory $HOME/Documents/MATLAB/dde_biftool_v3.2a/demo.

(iii) Create a new sub-directory Hutchinson.

(iv) Enter the sub-directory and create a file gen_sym_Hutchinson.m and use Listing 1 to create the system file for the delay differential equation (2).

(v) Create a new file Hutchinson.m. This file should contain the code from the items below.

(vi) Load the various directories of DDE-BifTool into the work-space and initialize the func structure, see Listing 2.

(vii) Manually construct a Hopf point as in Listing 1.7.

(viii) Compute the first Lyapunov coefficient and compare the result with the analytically derived coefficient.

References

[1] C. V. Hollot and Y. Chait. Nonlinear stability analysis for a class of tcp/aqm networks. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, volume 3, pages 2309–2314, Dec 2001.

[2] S.-I. Niculescu and K. Gu. *Advances in time-delay systems*, volume 38. Springer Science & Business Media, 2012.