SciComp Coursework 1

Max Oblein

mo17165@bristol.ac.uk

November 14, 2019

Contents

1 Summary of software **1**

2 Usage **3**

2.1 Shooting . . . 3

2.2 Natural_continuation . . . 4

2.3 Pseudo_continuation . . . 4

2.4 Demo file . . . 4

2.5 Test file . . . 5

3 Design decisions **5**

3.1 Shooting . . . 5

3.2 Numerical continuation . . . 5

4 Reflective learning log **6**

1 Summary of software

This package is designed to implement a numerical approach to solving boundary value ordinary differential equations (odes). More specifically, the software is designed for use on odes in which limit cycles are present. The methods used are numerical shooting to find a limit cycle and numerical continuation, both natural parameter and pseudo arc length, to find branches of limit cycles.

The function `shooting` takes an input of an: ode system, its parameters, and a guess at the initial conditions and period of oscillation of the system. The program then conducts a shooting step to correct these guesses subject to the constraints

$$U(0) = U(T) = 0 \tag{1}$$
$$U_0'(t) = 0, \tag{2}$$

where equation 1 forces the shooting code to find a limit cycle with period T and equation 2 is a phase condition to add another constraint. The function is solving for two unknowns so there must be two constraints and for almost all ode systems equation 2 is an adequate phase condition.

The function `natural_continuation` implements a natural parameter continuation to find a branch of limit cycles within an ode system. It finds the appropriate initial conditions for a limit cycle with a fixed parameter that is incremented by a fixed step size each iteration. The next limit cycle is found by calling

$$\texttt{shooting}(\texttt{odefunc}, \texttt{params}, \texttt{U0}), \tag{3}$$

where `params` is a tuple of parameter with the varying parameter incremented by the step size Δ and `U0` being the conditions of the previous limit cycle. This is a good initial guess as Δ is small.

The demo file provided with the package shows this being performed on the Hopf bifurcation ode system, as seen below in Figure 1. The function also allows for a discretisation method to be chosen. In most cases for odes this is `shooting`. However, this can be changed to a lambda function that returns U each step; which allows the continuation method to be performed on polynomial equations rather than just odes to plot the roots of the polynomial as a parameter is changed.

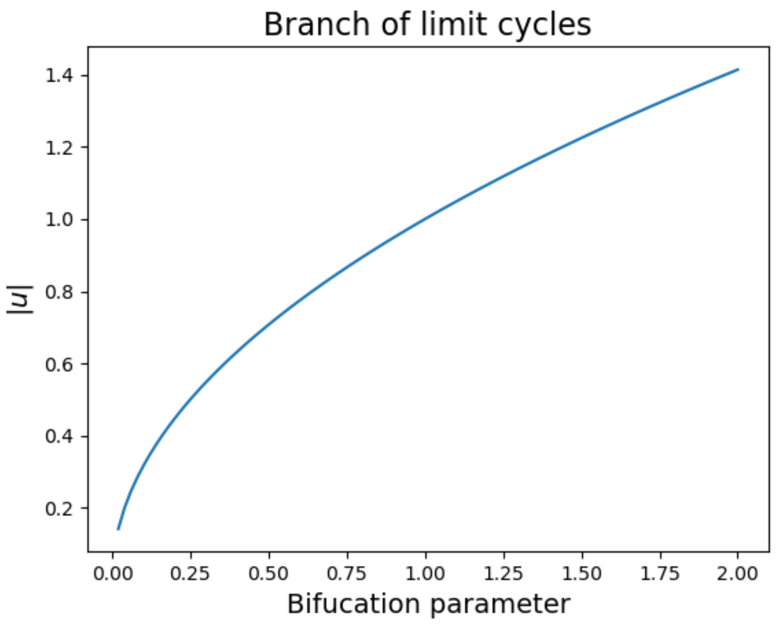

Figure 1: Graph showing a branch of limit cycles in the Hopf bifurcation system found using natural parameter continuation

The function `pseudo_continuation` is another function for numerical continuation. However, in this case a pseudo arc length approach is used. This is to avoid an issue with natural parameter continuation where when a bifurcation curve shows a fold the natural parameter continuation fails. Rather than finding the entire branch of limit cycles the algorithm just finds the nearest steady state. To approach this problem the parameter is now allowed to change each iteration, unlike in natural parameter continuation. To make sure the algorithm finds the folded branch the constraint 6 is added.

When initialising this method two vectors v_0 and v_1 which contain the state variables, the period of the limit cycle and the parameter for that cycle, are created. The guess at v_2 is then found by

$$\tilde{v_2} = v_1 + dv, \tag{4}$$

where

$$dv = v_1 - v_0. \tag{5}$$

This guess is then corrected by allowing the parameter to change and searching for the limit cycle branch along a line orthogonal to dv by adding in the constraint

$$\tilde{v_2} \cdot dv = 0. \tag{6}$$

In the demo file provided this function is run on the modified Hopf bifurcation ode system and the resulting graph is seen below in Figure 2.

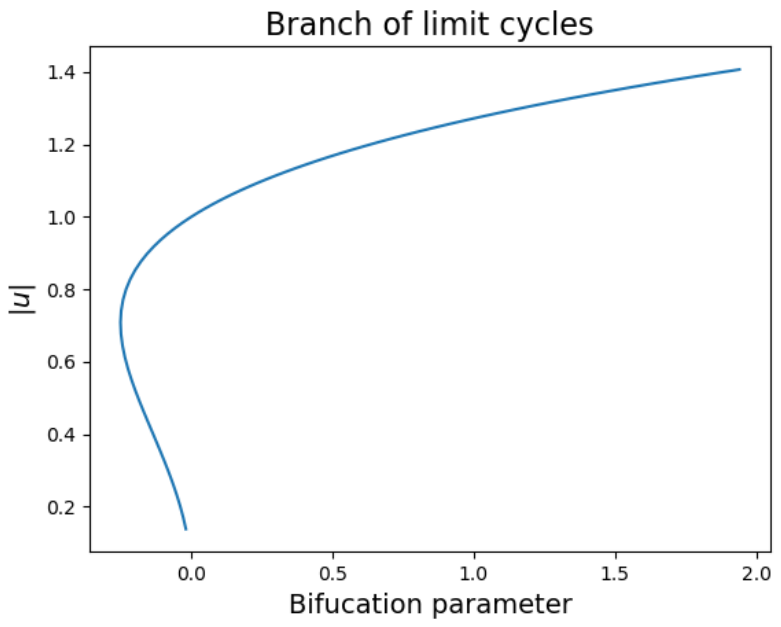

Figure 2: Graph showing a branch of limit cycles in the modified Hopf bifurcation system found using pseudo arclength continuation

2 Usage

2.1 Shooting

Inputs	
odefunc	System of first order odes as used for sciPy's `odeint`.
params	The parameters of the ode provided in a tuple.
v	Initial guess at state variable initial conditions and the time period of the limit cycle to be found.
Outputs	
solution	An array of the corrected initial conditions and period of the nearest limit cycle.

2.2 Natural_continuation

Inputs	
u0	Guess at initial state variables and time period of limit cycle.
params	The parameters of the ode provided in an array with one element being a range for that parameter to vary through.
odefunc	System of first order odes as for sciPy's `odeint`.
vary_param	Index at which the varied parameter lies, defaults to 0.
steps	Integer value for the number of steps within parameter range, defaults to 100.
discretisation	Method to be used. For odes use shooting and for polynomials use default lambda.
plot	Boolean option as to whether to show a plot of the branch of limit cycles, defaults to False.
Outputs	
sol_list	An array of the limit cycle conditions (initial state variables and period) with their parameters.

2.3 Pseudo_continuation

Inputs	
u0	Guess at initial state variables and time period of limit cycle.
params	The parameters of the ode provided in an array with one element being a range for that parameter to vary through.
odefunc	System of first order odes as for sciPy's `odeint`.
vary_param	Index at which the varied parameter lies, defaults to 0.
steps	Integer value for the number of steps within parameter range, defaults to 100.
discretisation	Method to be used. For odes use shooting and for polynomials use default lambda.
plot	Boolean option as to whether to show a plot of the branch of limit cycles, defaults to False.
Outputs	
sol_list	An array of the limit cycle conditions (initial state variables and period) with their parameters.

2.4 Demo file

Within the repository there is a demo file `demo.py` this is a very short file that allows the user to see some basic usage of the functions and plot Figures 1 and 2 as seen in this report. To access these simple demos the user should call the `demo.py` file on the command line with an option for which demo to use, either `natural` for Figure 1 or `pseudo` for Figure 2.

2.5 Test file

Also included with the package is a very simple test file which runs some tests for the outputs of both shooting and continuation and one to check that the code errors if given an incorrect input. Test one is to check the output of the `shooting` function this will shoot with the Hopf bifurcation ode system, a set of parameters and an initial guess. The test will then compare the output limit cycle to the output of the analytical solution to the equations, if it is sufficiently close the test is passed. The next test is to test the output of `natural_continuation`. Using the Hopf bifurcation ode system it can be found from the analytical solution that the branch of $|U|$ vs β should be equal to $\sqrt{\beta}$ if the solutions are sufficiently close this test passes. There is also a test to check that the dimensions of the inputted $U0$ are equal the dimensions taken by the ode function. This test will run the continuation code with an incorrect sized $U0$ and the function should print an error and return 1. Finally I have included a test to check that when running with an incorrect index for `vary_param` for example in the Hopf bifurcation ode you vary β but if you forget to input a `vary_param` it defaults to zero which indexes to σ so again an error should be returned for this test to pass.

3 Design decisions

3.1 Shooting

The first coding choice in creating a numerical shooting algorithm is to decide which ode solver to use. Initially I decided to use a Runge-Kutta fourth order method as I had coded one previously in this unit. However, after moving forward with the shooting code I decided to change to using sciPy's `odeint` function. This decision was made due to the increased speed of `odeint` and the generality built into sciPy, these factors combined with the extensive work done to develop `odeint` made it a better option for this package than my own function.

Another decision was whether to use my own root finding algorithm based on the Newton-Raphson method or whether to use sciPy's `fsolve`. I decided to use `fsolve` as it is highly optimised by all the contributors to the library and therefore will run much faster than any function I could write during the short time scale of this project. It also makes good decisions as to when to terminate for a tolerance with an easy to use interface. All this makes `fsolve` the obvious choice for this project.

Originally rather than using a phase condition to solve for the time period in the shooting code I had written a function that used sciPy's function for finding all the peaks within a signal, to find the peaks of the ode solution and calculate a period from this. The `shooting` code then optimises the initial conditions for a limit cycle with this period. However this has a higher computational cost than just adding another constraint or phase condition to solve for the period as well.

When solving for the period, initially the phase condition was an argument passed to the shooting function. However as the project went on all of the ode systems tested used the phase condition given in equation 2, therefore I removed the argument to streamline usage of the code whilst still achieving adequate results on the given odes.

3.2 Numerical continuation

The main decision within this section was the addition of a optional boolean argument for plotting. The branch of limit cycles found through numerical continuation is outputted as an array but this is rather hard to understand. Personally I found it much easier to understand the bifurcation when plotting the diagram as seen in both Figures 1 and 2. This option however can cause issues running the code as it halts any progress until the figure is closed. When looking at the functions in common libraries such as sciPy they all output arrays for the user to plot with later however an optional argument to plot seems to be the best of both worlds.

Another decision was the inclusion of a discretisation optional argument. This allows for the user to decide to use `shooting` for finding limit cycles in periodic odes and the use of a lambda function for finding the roots of polynomials where shooting on to a solution is not needed as limit cycles do not exist, we are just searching for the roots of the polynomial given a parameter.

4 Reflective learning log

The main take away from this project for me is the planning involved in writing usable software. At the start of the project I was writing functions that were designed to achieved a short term specific goal, however as the project went on I had to go back and generalise those functions to work in other cases. With some more careful planning a lot of time could have been saved by writing general functions from the start.

Secondly I learnt the importance of using version control software. When writing this package there were a few times that when attempting something new I ended up breaking my previous code. This could have been avoided with more careful structuring of the modules in my code and hopefully that will be the case in the future. However when these issues occurred during this project I could use git to easily revert to a previous commit where the code was working correctly.

After starting this unit and learning to use git for version control I made sure that my current MDM project makes full use of the collaborative power of using an online GitHub repository to share and edit our code. This meant we can all see any edits made to the project at regular intervals. Also in the MDM project I have tried to plan and generalise my code so that it is usable for the rest of the members of my group.

The mathematical content of the project has a lot of crossover with that of the Non-linear dynamics and chaos course. The study of limit cycles and bifurcation diagrams has given me further understanding of the systems being studied in NDC helping to reinforce my learning.

As far as the numerical methods used in this project go I have learnt how to solve periodic boundary value problems to find the limit cycles within the system. This could definitely be very useful in an MDM project as very often a mathematical model is heavily based on a system of odes. This package may be reused in the future to allow me to find any limit cycles needed for future projects.