Numerical continuation of limit cycles in large systems in MATLAB

Jonkhout, C.J.H.

C.J.H. Jonkhout

Numerical Continuation of Limit Cycles in Large Systems with MATLAB

Master thesis

Supervisors:
Yuri Kuznetsov (Utrecht University and University of Twente)
Mark Pekkér (University of Alabama in Huntsville)
Arjen Doelman (University of Leiden)

Date Master exam: 2019

Mathematical Institute, University of Leiden

Contents

Chapter 1

Introduction

The study of systems of ODEs of one hundred or more equations is becoming more and more common. Such systems can arise by discretizing a PDE in its spatial dimensions. Large systems are also seen in systems biology and neurophysiology, where large systems of ODEs are used to describe the complex dynamics of various biological processes. Periodic orbits can occur in both types of large systems of ODEs. For example, in [1] it is described that sustained oscillations were seen in a model of the mitochondrial respiratory chain.

Therefore, efficient methods for investigating periodic orbits in large systems of ODEs are needed. In this thesis we focus on the dependence of periodic orbits on parameters in large systems of ODEs, by computing the changes in a periodic orbit that occur if a parameter is changed. The process of tracking changes in cycles caused by a change of a parameter is called continuation of cycles. Continuation can also be applied to equilibria and their bifurcations, as well as to homoclinic and heteroclinic orbits. In section 1.2 a rigorous definition of continuation is given, and a basic algorithm for continuation is described.

In the software package Matcont, numerical continuation of limit cycles and their bifurcations in small systems of ODEs based on orthogonal collocation is implemented. Orthogonal collocation is also known as Gauss-Legendre collocation or spline collocation. Using Matcont, limit cycles could already be continued in systems of first order ODEs with up to about 100 equations. However, the methods for detection of bifurcations used in Matcont do not perform well for large systems. The cycle routines from Matcont were imported into the software package CL_MATCONTL and subsequently modified to suit large systems, as a part of this thesis project. In chapter 2, we discuss the changes we made in this code that allow continuation of cycles in systems of up to about 400-500 equations, and in chapter 5, we discuss methods for detecting bifurcations that scale well up to any size system in which cycles can be continued.

In addition to improving in the orthogonal collocation code, the methods single shooting with Newton-Picard, and multiple shooting with Newton-Picard were implemented.[1] The shooting methods will be discussed in their respective chapters.

Three examples that illustrate the methods that were implemented are presented in chapter 6. The first example is a 1D transport model of plasma in a TOKAMAK reactor. Using the orthogonal collocation method cycles have been continued in this system of three PDEs for the first time (see Figure 1). The Brusselator reaction-diffusion model is also discussed. It is a system in which it is very easy to vary the number of mesh points. Hence,

[1] K. Lust mentions in his thesis that orthogonal collocation with Newton Picard is also possible, but since he did not provide a complete mathematical description of such a method, it is outside the scope of this thesis.

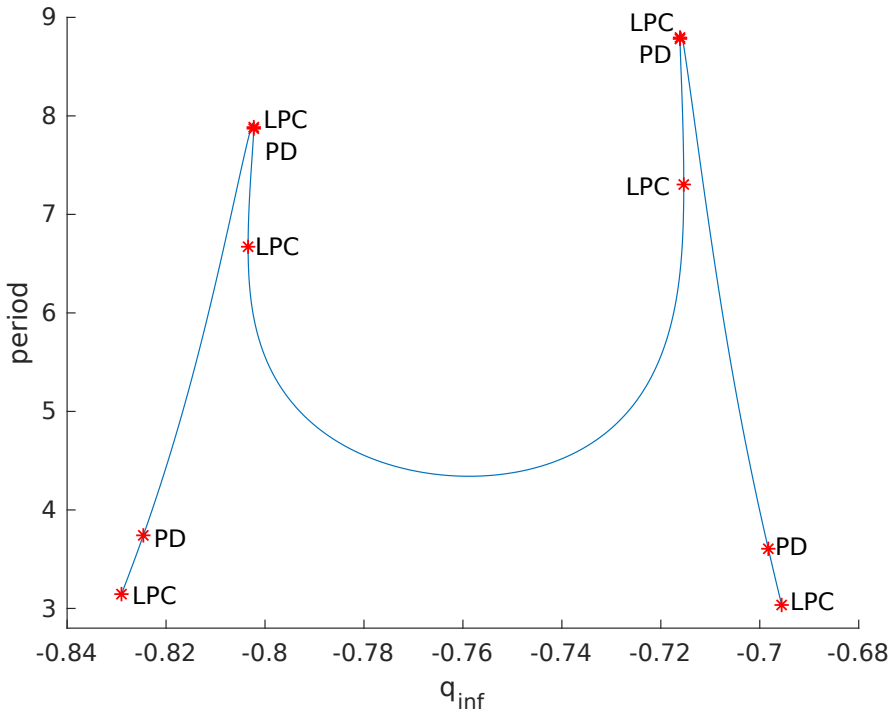


Figure 1.1: Continuation of a cycle in the plasma-system for $N = 50$

it is an good system to demonstrate to capabilities of the Newton-Picard methods, which are especially suited to very large systems. In the third and last example, a branch of cycles is continued in a convection-diffusion system that models a nonadiabatic tubular reactor.

The various continuation methods for cycles are features that were added to CL_MATCONTL, which is a continuation package for bifurcations in large systems of ODEs. CL_MATCONTL is derived from Matcont, which is a continuation package for bifurcations for ODEs [2]. CL_MATCONTL already supported continuation of equilibria, and many bifurcations of equilibria using the continuation of invariant subspaces [3]. However, continuation of cycles was not yet supported. A collection of tutorials for CL_MATCONTL is available on the website of Mark Pekkér (aka Mark Friedman) [4], which will include several tutorials about cycles based on this thesis.

The main strategy used in CL_MATCONTL for efficient continuation of bifurcations of equilibria is continuation of invariant subspaces. The Newton-Picard methods for cycle continuation also use invariant subspaces. The main difference between the use of invariant subspaces for equilibria and the use of invariant subspaces for cycles is that the matrix whose action preserves the subspace is fully available during continuation of equilibria, but is not readily available during continuation of cycles.

One important aspect of Newton-Picard methods, is that unlike all the previous types of continuation curves in Matcont and CL_MATCONTL, the Newton-Picard method does not use the `newtcorrL` function for corrections. Due particular nature of the Newton-Picard method, it is more practical to write a separate corrector function for each Newton-Picard method. In chapters 3 and 4 in is stated where to find these functions in the CL_MATCONTL code.

1.1 Summary of contributions to CL_MATCONTL

In summary, the following contributions were made to CL_MATCONTL as a part of this thesis:

- The code for continuation by orthogonal collocation was taken from Matcont, and merged into CL_MATCONTL
- The orthogonal collocation code was improved and adapted for large systems. (see section 2.1)
- The curve-files `single_shooting.m` and `multiple_shooting.m` were added. Curve-files specify the continuation curve by defining a function F as in algorithm 1
- Newton-Picard methods for single shooting and multiple shooting were added
- Methods for detection of bifurcations were conceived and implemented which can cope with large systems.

1.2 Continuation

Continuation is a process of finding a curve of solutions to a parameter-dependent problem. Specifically, suppose F is a continuously differentiable function from $\mathbb{R}^n \times \mathbb{R}$ to $\mathbb{R}^n$, we know $F(y_0, \gamma_0) = 0$ for a given $y_0 \in \mathbb{R}^n$ and $\gamma_0 \in \mathbb{R}$, and we wish to find an approximation to the curve $(y(\eta), \gamma(\eta))$, parameterized by η, for which $F(y(\eta), \gamma(\eta)) = 0$, $y(0) = y_0$, $\gamma(0) = \gamma_0$. The most obvious example of a continuation problem is the continuation of an equilibrium of a system of autonomous first order ODEs $dy/dt = f(y, \gamma)$, although continuation can be applied to other problems as well, such as the continuation of limit cycles.

To get an idea of what a continuation procedure look like, see Algorithm 1. Note that in practice a continuation procedure would include many refinements, which are not shown there. One of the most important refinements is adaptation of the step size. This is, in a limited sense, comparable to step size adaption in time integration of ODEs. In both cases one can achieve optimal performance by adjusting the step size after each step, based on the properties of the curve at the current step. In continuation, an important step size adjustment is to reduce the step size if the Newton corrections fail to converge or to increase it if the corrections converge rapidly.

Input:

- a continuously differentiable function
  $F : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n : (y, \gamma) \mapsto F(y, \gamma)$
- an initial point $(y_0, \gamma_0) \in \mathbb{R}^n \times \mathbb{R}$ such that $\left\|F(y_0, \gamma_0)\right\| < tolerance_F$.
- $stepsize > 0$, and $tolerance_F$, $tolerance_{\Delta x} > 0$

Result: A list of approximations to points
$(x_0, ..., x_m) = ((y_0, \gamma_0), ..., (y_m, \gamma_m)) \in \mathbb{R}^n \times \mathbb{R}$ on the curve defined by $F(y, \gamma) = 0$

Set $i \leftarrow 0$

while $i < m$ **do**

- Compute the Jacobian matrix $F_{x,\gamma}(x_i, \gamma_i) = \frac{\partial(F)}{\partial(x,\gamma)}(x_i, \gamma_i)$. This can be done either by finite differences, or evaluating the symbolic derivative.
- Compute a vector $v_i^* \in \mathbb{R}^{n+1}$ that spans the kernel of $F_{x,\gamma}(x_i, \gamma_i)$. The vector v_i^* is tangent to the curve we wish to approximate.
- Normalize v_i^* by computing $v_i \leftarrow v_i^* / \left\|v_i^*\right\|$
- Predict the next point: $x_{i+1} = x_i + stepsize \cdot v_i$
- Apply Newton corrections until $\left\|F(x_{i+1})\right\| < tolerance_F$ and the norm of the last correction Δx is less than $tolerance_{\Delta x}$

end

Algorithm 1: Continuation

The Newton Corrections are defined as follows. To obtain a unique solution of $F(x) = 0$ near x_{i+1}, we require that x_{i+1} satisfies:

$$\langle v_i,\ x_{i+1} - x_i \rangle = 0$$

This method of defining the next point in the continuation is called the pseudo-arclength continuation [5, section on pseudo-arclength continuation].

Therefore, to solve for x_{i+1} we apply Newton's method to

$$\begin{cases} F(x_{i+1}) = 0 \\ \langle v_i,\ x_{i+1} - x_i \rangle = 0 \end{cases}$$

Suppose we have a candidate x_{i+1}^j for the next point x_{i+1} in the continuation. We compute an approximation x_{i+1}^{j+1} that is more accurate than x_{i+1}^j by computing the Jacobian matrix $\partial_x F(x_{i+1}^j)$, and then solving

$$\begin{pmatrix} \partial_x F(x_{i+1}^j) \\ v_i^T \end{pmatrix} (x_{i+1}^{j+1} - x_{i+1}^j) + \begin{pmatrix} F(x_{i+1}^j) \\ 0 \end{pmatrix} = 0$$

where we see v as a column vector. Thus, Newton Corrections are performed as follows:

Input:

- an approximation x to a point on the curve $F(x) = 0$
- a tangent vector v (as a column vector) to the curve $F(x) = 0$ at a previous continuation point
- tolerances $tolerance_F$ and $tolerance_{\Delta x}$

Output:

- an approximation x such that $\left\|F(x)\right\| < tolerance_F$ and $\left\|\begin{pmatrix}\partial_x F(x)\\ v^T\end{pmatrix}^{-1} F(x)\right\| < tolerance_{\Delta x}$

done $\leftarrow$ false
while *not done* **do**
  $\Delta x \leftarrow -\begin{pmatrix}\partial_x F(x)\\ v^T\end{pmatrix}^{-1} F(x)$
  $x \leftarrow x + \Delta x$
  done $= \left\|F(x)\right\| < tolerance_F$ and $\left\|\Delta x\right\| < tolerance_{\Delta x}$
end

Algorithm 2: Newton Corrections

Note that we drop the subscript and superscript of x here, since we don't need them. Once the approximation starts to converge, one should see quadratic convergence, that is $\left\|F(x_i^{j+1})\right\| \approx \left\|F(x_i^j)\right\|^2$, until other numerical errors become dominant. Since we are interested in the x rather than $F(x)$, it is important to demand that the norm of Δx is below a certain tolerance, since if the gradient of $F(x)$ is small, there will be large changes in x even if $F(x)$ is nearly zero.

Chapter 2

Continuation of cycles using orthogonal collocation

Orthogonal Collocation is a method of discretizing a Boundary Value Problem for ODEs [6]. Continuation of a limit cycle can be viewed a boundary value problem. Hence, the problem of continuing a limit cycle can be discretized using orthogonal collocation [7]. Specifically, suppose we wish to continue limit cycles in the system:

$$dy/dt = f(y, \gamma), \quad y \in \mathbb{R}^n, \quad \gamma, t \in \mathbb{R}$$

and let $\phi(y_0, t, \gamma)$ be the solution of the initial value problem $d\phi/dt = f(\phi, \gamma)$, $\phi(0) = y_0$, evaluated at time t. The boundary value problem we must solve to find a limit cycle is

$$\begin{cases} d\phi/dt = f(\phi, \gamma) \\ \phi(y_0, \gamma, 0) = \phi(y_0, \gamma, T) \end{cases} \tag{2.1}$$

One would solve this problem for a fixed γ by finding an initial value $y_0 \in \mathbb{R}^n$, and a period $T \in \mathbb{R}$ such that the equations (2.1) are satisfied.

It is customary to rescale time such that the period of the cycle in the rescaled time is one. With this rescaling (2.1) becomes

$$\begin{cases} d\phi/dt = Tf(\phi, \gamma) \\ \phi(y_0, \gamma, 0) = \phi(y_0, \gamma, 1) \end{cases} \tag{2.2}$$

Suppose that a cycle C with period T has been found. Note that any point y_0 on C will satisfy (2.2). To uniquely define y_0 we must add a phase condition to (2.2). In a continuation context this is done using the derivative w.r.t. time of the periodic solution w at the previous continuation step. Hence, the phase condition will be

$$\int_0^1 \langle \dot{w}(\tau), \phi(\tau) \rangle d\tau = 0 \tag{2.3}$$

This phase condition ensures that ϕ is close to w. Specifically, phase condition ensures that the ϕ is a minimizer of the 2-norm of the difference of ϕ and w, when minimizing over the set of functions $\{\phi_s : \tau \mapsto \phi(\tau + s) \quad | \quad s \in [0, 1]\}$, i.e. the set of all phase shifts of ϕ. [5].

When using orthogonal collocation, we approximate the periodic orbit with a piecewise polynomial function. The periodic orbit is approximated using n_{tst} polynomials. Each

polynomial approximates the periodic orbit on one of the n_{tst} mesh intervals. We denote the boundaries of these intervals by:

$$0 = \tau_0 < \tau_1 < \cdots < \tau_{n_{tst}} = 1 \tag{2.4}$$

We will call the points (2.4) mesh points. The mesh points can be chosen freely within the constraints of (2.4). They are typically chosen to optimize the accuracy of the discretization.

On each mesh interval $[\tau_i, \tau_{i+1}]$ for $i \in \{0, ..., n_{tst} - 1\}$ we define basis points $\tau_{i,j}$:

$$\tau_{i,j} = \tau_i + \frac{j}{n_{col}}(\tau_{i+1} - \tau_i) \quad \text{for} \quad j \in \{0, \ldots, n_{col}\} \tag{2.5}$$

Note that the last basis point of mesh interval i is equal to the first basis point of mesh interval $i+1$, that is, we have $u^{i,n_{col}} = u^{i+1,0}$ for $i \in \{1, ..., n_{tst}-1\}$. On each mesh interval $[\tau_i, \tau_{i+1}]$ for $i \in \{0, ..., n_{tst} - 1\}$ the solution will be approximated by the polynomial:

$$u^{(i)}(\tau) = \sum_{k=0, k\neq j}^{n_{col}} u^{i,j} l_{i,j}(\tau) \tag{2.6}$$

where $u^{i,j}$ are constants and $l_{i,j}$ are a Lagrange basis polynomials, defined as

$$l_{i,j}(\tau) = \prod_{k=0, k\neq j}^{n_{col}} \frac{\tau - \tau_{i,k}}{\tau_{i,j} - \tau_{i,k}}$$

Note that we have

$$l_{i,j}(\tau_{i,j}) = \prod_{k=0, k\neq j}^{n_{col}} \frac{\tau_{i,j} - \tau_{i,k}}{\tau_{i,j} - \tau_{i,k}} = 1$$

and if $j^* \in \{0, ..., n_{col}\}$ and $j^* \neq j$ then we have

$$l_{i,j}(\tau_{i,j^*}) = \prod_{k=0, k\neq j}^{n_{col}} \frac{\tau_{i,j^*} - \tau_{i,k}}{\tau_{i,j} - \tau_{i,k}} = 0$$

since one of the factors of the numerator vanishes. Thus, we have $u^{(i)}(\tau_{i,j}) = u^{i,j}$. Hence, we conveniently choose the points $\tau_{i,j}$, as the points where the ϕ equals $u^{(i)}$ exactly, since then the coefficients $u^{i,j}$ are simply equal to $\phi(\tau_{i,j})$. Consequently, to compute the first point on a continuation curve of cycles, we find $u^{i,j}$ by computing $\phi(\tau_{i,j})$. Here ϕ is typically found by time integration towards a stable cycle, or constructing a small, elliptic orbit around a Hopf point.

Now, we require that the periodic orbit is a solution of the differential equation. When discretizing using orthogonal collocation, we require that the differential equation is satisfied at the Gauss points $\zeta_{i,k}$ in each mesh interval. The Gauss points are the roots of the n_{col}-th order Legendre polynomial relative to the mesh interval. Specifically, suppose η_k is the k-th root of te m-th order Legendre polynomial. Then we have:

$$\zeta_{i,k} = \tau_i + \frac{\eta_k + 1}{2}(\tau_{i+1} - \tau_i) \quad \text{for} \quad i \in \{0, \ldots, n_{tst} - 1\} \quad \text{and} \quad k \in \{1, \ldots, m\}$$

Note that the roots of the Legendre polynomial lie in the interval $(-1, 1)$. Hence, we have $\zeta_{i,k} \in (\tau_i, \tau_{i+1})$ for all $i \in \{0, \ldots, n_{tst} - 1\}$ and $k \in \{1, \ldots, n_{col}\}$.

Having defined the Gauss points, we can now require that the differential equation is satisfied at these points:

$$\sum_{j=0}^{n_{col}} u^{i,j} l'(\zeta_{i,k}) = Tf\left(\sum_{j=0}^{n_{col}} u^{i,j} l(\zeta_{i,k}), \gamma\right) \quad \text{for} \quad i \in \{0,\ldots,n_{tst}-1\} \quad \text{and} \quad k \in \{1,\ldots,n_{col}\} \tag{2.7}$$

The discrete version of the condition $\phi(y_0, \gamma, 0) = \phi(y_0, \gamma, 1)$ is:

$$u^{0,0} = u^{n_{tst}-1,n_{col}} \tag{2.8}$$

Finally, the discrete version of (2.3) is then

$$\sum_{i=0}^{n_{tst}-1} (\Delta t)_i \sum_{j=0}^{n_{col}} \omega_j u^{i,j} Tf(v^{i,j}, \gamma) = 0 \tag{2.9}$$

where $(\Delta t)_i$ is the width of the i-th mesh interval, and ω_j are the Newton-Cotes quadrature coefficients of degree $n_{col} + 1$.

2.1 Improvements made to cycle continuation in Matcont

The algorithm described in the first part of this chapter, has been implemented in Matcont since 2002 [8, 9]. The Matcont implementation was based on the collocation code in AUTO86 [10]. We will describe the changes made to the Matcont implementation of continuation of cycles by collocation to allow the continuation of cycles in large systems of ODEs.

First of all, the computation of the initial tangent vector for continuation as it was implemented in Matcont, was not well suited for large systems. The old way of computing the tangent vector sometimes required multiple attempts to find the tangent vector. As the size of the system increased, the number of attempts needed to be increased as well. Sometimes the procedure would fail all together.

A better way to compute the tangent vector is to pick a vector from the nullspace of the Jacobian matrix of the defining function of the curve of cycles.

Secondly, the computation of the Jacobian matrix of the curve function had to be made more efficient in terms of memory footprint. Since this Jacobian matrix is large and sparse, it should be stored as a sparse matrix, which was to a certain degree already the case in Matcont. One important improvement is that a more accurate upper bound on the number of nonzero's in the matrix is specified.

Finally, the way Floquet multipliers are computed has also been improved. The computation time has been reduced.

2.1.1 Sparsity of the Jacobian matrix

As mentioned, the Jacobian matrix is large and sparse. An important improvement in CL_MATCONTL was to derive a more accurate upper bound on the number of nonzero's of this matrix. This is important, since for a large enough system, one will run out of memory. Hence, with this improvement larger systems can be continued with the same amount of memory. One should note that CL_MATCONTL leaks memory. That is, during continuations, the amount of memory used by Matlab will steadily increase. For continuations of large enough systems, this is problematic. These memory leaks could are

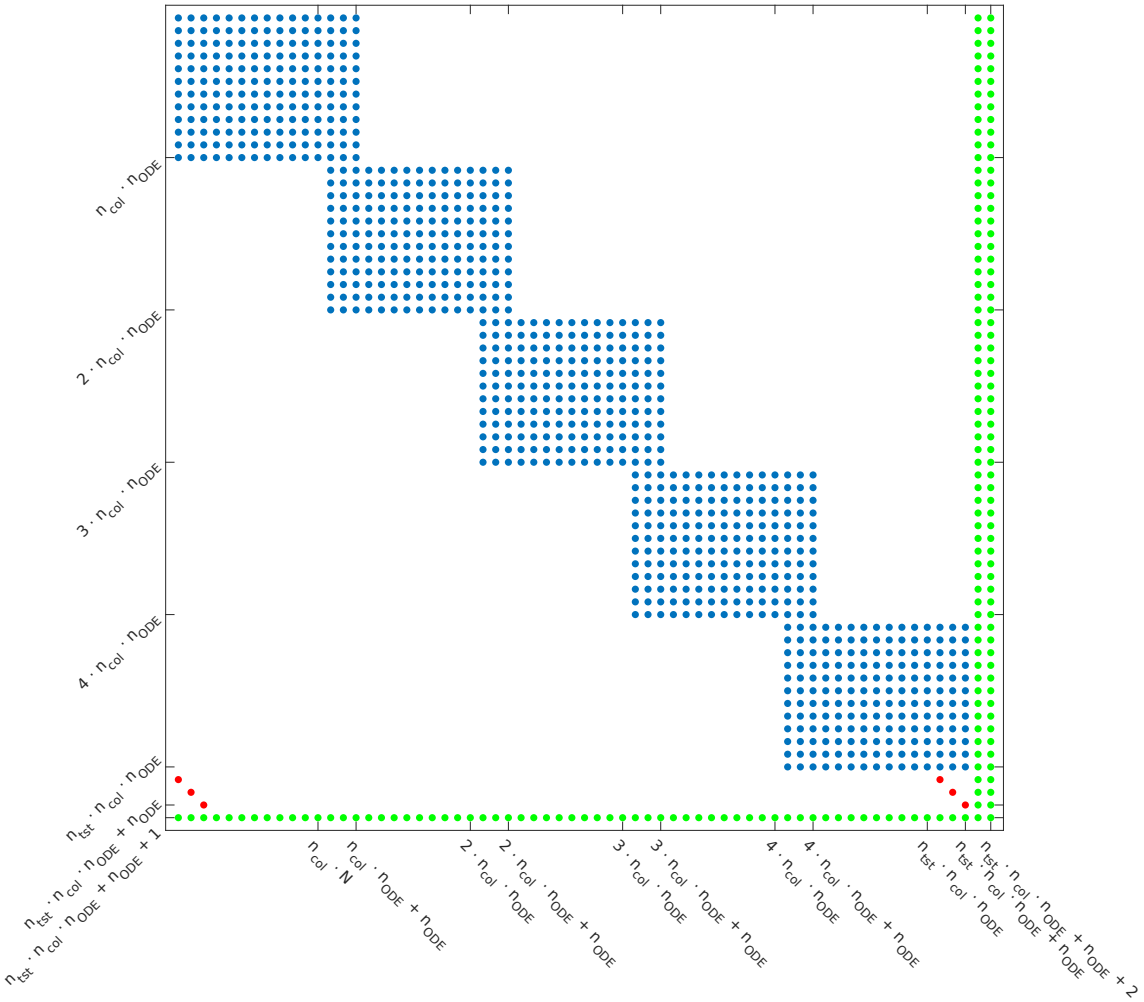


Figure 2.1: Sparsity pattern of the Jacobian matrix

bugs in either CL_MATCONTL or Matlab, and, ideally, they should be fixed.[1] In the meantime, the easiest way deal with these memory leaks is to restart Matlab often, and to use the latest version of Matlab.

The part of the code where the reduction of memory use was realized, were the mex-functions in CL_MATCONTL. Mex-functions are extensions to Matlab written in C or Fortran. The mex-functions in Matcont compute the Jacobian matrix of the limit cycle continuation problem, and other similar Jacobian matrices related to period doubling bifurcations (PD), limit points of cycles (LPC), Neimark-Sacker bifurcations (NS), and Branching Points of Cycles (BPC). These Jacobian matrices are sparse, and very large. The reason that the effort was made to write the functions that compute these Jacobian matrices as a mex-function, was to speed up the continuation of cycles (see the last section of [11]).

The sparsity structure of the Jacobian matrix of the cycle continuation problem is illustrated in the diagram below. Note that some nonzero's we count will be zeros for specific problems. Hence, the number of nonzero's we count now represents the worst case. In this diagram the number of coordinates of the system of ODE's n_{ODE} is three, the number of mesh intervals n_{tst} is five, and the number of collocation points n_{col} is four.

[1] perhaps by using http://undocumentedmatlab.com/blog/undocumented-profiler-options

We count the number of non-zero elements as a function of n_{ODE}, n_{tst}, and n_{col}. The height of the matrix is $n_{tst}\, n_{col}\, n_{ODE} + n_{ODE} + 1$, and the width of the matrix is $n_{tst}\, n_{col}\, n_{ODE} + n_{ODE} + 2$. The rectangular blocks (colored in blue in the diagram) are $n_{col}\, n_{ODE}$ elements high and $n_{col}\, n_{ODE} + n_{ODE} = (n_{col} + 1)n_{ODE}$ elements wide. Thus, the number of nonzero's in one blocks is $n_{col}\,(n_{col}+1)\, n_{ODE}^2$, and the number of nonzero's in all these blocks is $n_{tst}\, n_{col}\,(n_{col}+1)\, n_{ODE}^2$

The nonzero's related to the boundary conditions are colored in red in the diagram above. There are $2n_{ODE}$ such nonzero's. The remaining nonzero's are the nonzero's in the last two columns and the last row (colored green in the diagram). We assume that the matrix can be dense here. The number of nonzero's in the last two columns and the last two rows is twice the height plus once the width minus two, which equals $3(n_{tst}\, n_{col}\, n_{ODE} + n_{ODE}) + 2$.

All together there are at most:

$$n_{nz} = n_{tst}\, n_{col}\,(n_{col}+1)n_{ODE}^2 + 3n_{tst}\, n_{col}\, n_{ODE} + 5\, n_{ODE} + 2 \tag{2.10}$$

nonzero's in the Jacobian matrix of the continuation problem of limit cycles.

Before this new bound on the number of nonzero's was used, Matcont allocated storage for $n_{tst}^2\, n_{col}^2\, n_{ODE}^2$ nonzero's. Hence, using (2.10) the memory footprint of the Jacobian matrix has been reduced by factor of n_{tst}, to leading order of n_{tst}, n_{col}, and n_{ODE}.

Note that when continuing cycles in spatially discretized PDEs, these Jacobian matrices much sparser than stated in (2.10). Should the need for even more frugal use of memory arise, a good strategy would be to allocate a fraction of the upper bound (2.10), and then reallocate and copy the Jacobian matrix if the allocated space is full. This will probably work well in practice, since memory to memory copying is very fast. Another more sophisticated option would be to assume or prove that the additional sparsity in one block (referring the blocks are in blue the diagram) is similar for each block.

One should keep in mind that after construction of the Jacobian matrix, its LU decomposition will be computed, which will cause some fill-in, and thus require more memory. Hence, at some point, reducing the storage requirements for constructing the Jacobian matrix will not reduce the minimum memory requirement for continuation of cycles using orthogonal collocation.

2.1.2 Floquet Multipliers

The way Floquet multipliers are computed during continuations of cycles using orthogonal collocation was also improved. Floquet multipliers are computed using the Jacobian matrix discussed in the previous section, evaluated at point that results from the Newton corrections. When computing the Floquet multipliers, only a part of the Jacobian matrix is used. Specifically, only the nonzeros that are colored in blue in Figure 2.1.1 are used. We refer to this submatrix as J^*. This can be explained as follows. The Jacobian matrix represents the rate of change of the defining function, w.r.t. the curve variables. At convergence the norm of the defining function is close to zero. Therefore, when computing multipliers, we assume it to be exactly zero. We refer to this assumption as assumption A.

The monodromy matrix is the Jacobian of $u^{n_{tst}-1,n_{col}}$ w.r.t. $u^{0,0}$. Using the matrix J^* and assumption A, we can express the variation of $u^{n_{tst}-1,n_{col}}$, in terms of $u^{0,0}$. In effect we will compute the linear map $M : u^{0,0} \mapsto u^{n_{tst}-1,n_{col}}$ such that we have $J^*u = 0$, where u is the column vector whose first n_{ODE} elements are equal to the elements of $u^{0,0}$, the last n_{ODE} elements are equal to minus one times the elements of $u^{n_{tst}-1,n_{col}}$ and whose other elements are zero.

The first step in computing the multipliers is called "condensation of parameters". In this step many nonzero's in J^* are eliminated used Gaussian elimination. The resulting sparsity pattern is illustrated in figure 2.1.2 for $n_{ODE} = 3$, $n_{tst} = 5$, and $n_{col} = 4$. Condensation of parameters can by carried out for each block in parallel. The system of equations corresponding to the nonzero's that are colored red and green in figure 2.1.2 can be solved independently. This way we have eliminated all the variables that correspond to the internal points of mesh intervals. We define the sequences of matrices A_i, and B_i for $i \in \{1, ..., n_{tst}\}$ as indicated in figure 2.1.2.

We will now derive an equation for $u^{n_{tst}-1,n_{col}}$ in terms of $u^{0,0}$. To make the equation more readable we will write u_i for $u^{i,0}$ ($i \in \{0, ..., n_{tst} - 1\}$, and $u_{n_{tst}} = u^{n_{tst}-1,n_{col}}$. Using this notation we find $A_i u_{i-1} + B_i u_i = 0$ for $i = \{1, 2, ..., n_{tst}\}$ from J^* after condensation of parameters. Hence we have $u_i = -B_i^{-1} A_i u_{i-1}$ for $i = \{1, 2, ..., n_{tst}\}$. By applying this formula recursively we have:

$$u_{n_{tst}} = B_{n_{tst}}^{-1}\ A_{n_{tst}}\ \ B_{n_{tst}-1}^{-1}\ A_{n_{tst}-1} \ldots B_2^{-1}\ A_2\ \ B_1^{-1}\ A_1 u_0$$

Thus an approximation of the monodromy matrix is:

$$B_{n_{tst}}^{-1}\ A_{n_{tst}}\ \ B_{n_{tst}-1}^{-1}\ A_{n_{tst}-1} \ldots B_2^{-1}\ A_2\ \ B_1^{-1}\ A_1 \tag{2.11}$$

Periodic Schur decomposition

We have shown how to compute the monodromy matrix from the Jacobian. However, in [12] it is argued that direct computation of the eigenvalues of the monodromy matrix using (2.11), is numerically unstable, when large eigenvalues are present. In [12], it is proposed that a numerically more stable approach is to use the periodic Schur decomposition. The periodic Schur decomposition is derived in [13]. With the permission of Daniel Kressner, his implementation of the periodic Schur decomposition was included in CL_MATCONTL, to compute Floquet multipliers of cycles. Daniel Kressner described some of the details of his implementation in [14].

Kressner's implementation is indeed efficient. It is partly written in FORTRAN. However, the current implementation does not always converge. Hence, it is sometimes necessary to fall back on direct matrix multiplication. This is fine, as long as there are no large multipliers. In case of large multipliers, CL_MATCONTL will fall back on another way of computing multipliers, namely the one that was implemented in Matcont, which is also uses the same Jacobian matrix. This algorithm was based on the Floquet multiplier algorithm of AUTO97. In [12] it shown that the error in the trivial multiplier of the AUTO97 algorithm and the periodic Schur decomposition is similar in one example. The AUTO97 algorithm incorporates insights from [15]. Its current Matlab implementation is, however, much slower than either direct matrix multiplication or using the Kressner's implementation of the periodic Schur decomposition.

Alternatively, it is possible to compute multipliers using the periodic Schur decomposition, even if it has not fully converged. However, this has not been implemented.

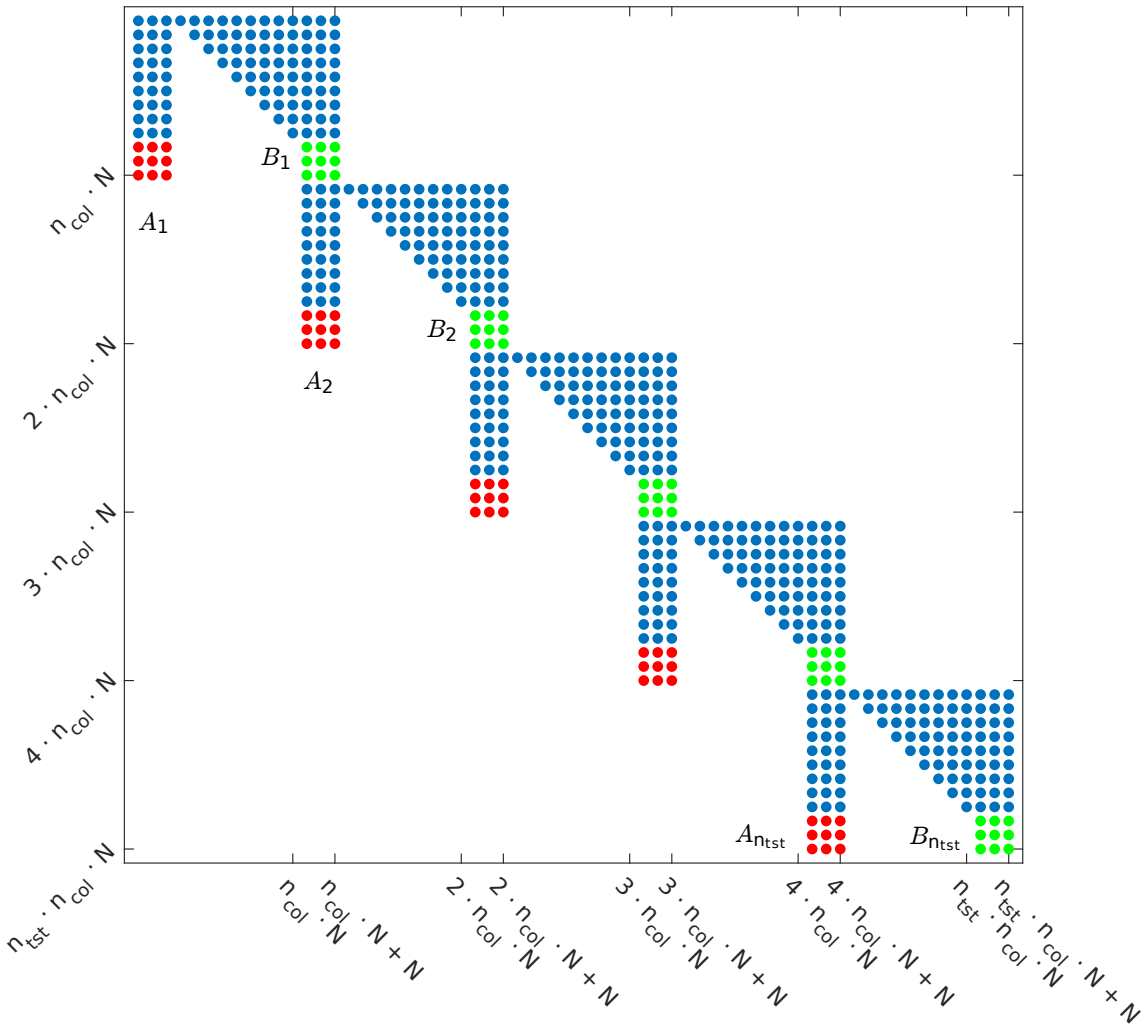


Figure 2.2: Sparsity pattern of J^* after condensation of parameters.

Chapter 3

Continuation of cycles using single shooting with Newton-Picard

We will describe a method of continuation of cycles of large systems of ODEs using the Newton-Picard method.

The Newton-Picard method described by K. Lust in [16, 17] and [18], is an algorithm for the continuation of cycles based on single shooting. First, we describe the standard method of continuing cycles with single shooting.

3.1 Continuing cycles with single shooting

We define a parameter-dependent autonomous system of first order ODEs:

$$\frac{dy}{dt} = f(y, \gamma), \quad y \in \mathbb{R}^n_{ODE}, \quad \gamma \in \mathbb{R}$$

To find a periodic orbit, we solve the system:

$$\begin{cases} r(y_0, T, \gamma) := \phi(y_0, T, \gamma) - y_0 & = 0 \\ s(y_0, T, \gamma) & = 0 \end{cases} \tag{3.1}$$

for y_0. Here T is the period, $\phi(y_0, T, \gamma)$ is the solution of the ODE for initial condition $y(0) = y_0$, evaluated at T, for parameter value γ, and s is a phase condition to make the solution unique. This condition is needed since to eliminate the invariance of a periodic solution under time translation. We use:

$$s(y_0;\ y_0^0, \gamma_0) = \left\langle\ f(y_0^0, \gamma_0), \quad y_0 - y_0^0\ \right\rangle \tag{3.2}$$

Where y_0^0 is the previous continuation point, and γ_0 is the parameters value at the previous continuation point. Hence, $f(y_0^0, \gamma_0)$ is the tangent vector to the periodic solution at y_0^0. Note that this is related to the phase condition used in continuation by orthogonal collocation. Specifically, in orthogonal collocation, one essentially integrates 3.2 over the entire cycle.

Once a solution is found, it can be continued by adding a pseudo arc-length continuation condition n to system (3.1). If we have $x = (y_0, T, \gamma)$, and v a tangent vector to the continuation curve at the previous continuation point x^*, and h is the step size, then the

continuation condition is $n(y_0, T, \gamma) = n(x) = \langle x - (x^* + hv), v \rangle$.

$$\begin{cases} r(y_0, T, \gamma) & = 0 \\ s(y_0, T, \gamma) & = 0 \\ n(y_0, T, \gamma) & = 0 \end{cases} \tag{3.3}$$

To solve system (3.3), Newton's method can be applied, as explained in section 1.2. At each continuation point the next continuation point is found by applying Newton corrections as follows. We compute the corrections $(\Delta y, \Delta T, \Delta\gamma)$ from

$$\begin{bmatrix} M - I & b_T & b_\gamma \\ c_s^T & d_{s,T} & d_{s,\gamma} \\ c_n^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} r(y_0, T, \gamma) \\ s(y_0, T, \gamma) \\ n(y_0, T, \gamma) \end{bmatrix} \tag{3.4}$$

Where

$$\begin{bmatrix} M - I & b_T & b_\gamma \\ c_s^T & d_{s,T} & d_{s,\gamma} \\ c_n^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} := \left. \frac{\partial(r, s, n)(y_0, T, \gamma)}{(y_0, T, \gamma)} \right|_{(y_0, T, \gamma)} \tag{3.5}$$

Then the corrections are applied:

$$y^{j+1} = y^j + \Delta y$$

$$T^{j+1} = T^j + \Delta T$$

$$\gamma^{j+1} = \gamma^j + \Delta\gamma$$

Specifically, for the choices for n and s we make here, we have:

$$\begin{bmatrix} M - I & b_T & b_\gamma \\ c_s^T & d_{s,T} & d_{s,\gamma} \\ c_n^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} = \begin{bmatrix} M - I & f(y_0, \gamma) & b_\gamma \\ f(y_0^0, \gamma_0)^T & 0 & 0 \\ v_y^T & v_T & v_\gamma \end{bmatrix}$$

Note that the vector v^T, which is the continuation tangent vector (see section 1.2), spans the entire last row of this matrix. Hence, v_y, v_T, and v_γ are the components of v that are related to y, T, and γ respectively. The matrix M is the monodromy matrix. It can be computed by finite differences, or by solving the variational problem:

$$\begin{cases} M'(t) = \partial_y f(\phi(t), \gamma)\, M(t) \\ M(0) = I \end{cases}$$

If the dimension n_{ODE} of the state vector of the ODE is large, computing the monodromy matrix M, by finite differences or by solving the variational problem will be prohibitively expensive in terms of computation time. If the ODE is a discretization of a PDE, the monodromy matrix will typically have many eigenvalues that are nearly zero. This is exploited by the Newton-Picard method.

3.2 Single Shooting with Newton-Picard

As mentioned in at the end of the previous subsection, straightforward continuation of cycles with single shooting, will be prohibitively expensive. The Newton-Picard method computes an invariant subspace V of M spanned by eigenvectors associated to the eigenvalues of with a norm greater than ρ. For the solution of the Q-systems to converge one

needs $\rho < 1$. The smaller ρ the faster the solution of the Q-system converges, but for small ρ the system must be solved in the Newton correction is larger. One could, for instance, choose $\rho = 1/2$. In practice, one should choose a value ρ to minimize total computation time.

The p dimensional invariant subspace V will be defined using the eigenvectors associated to the dominant eigenvalues of M. Suppose we have an orthogonal basis V_p for V. Then we can decompose $\mathbb{R}^n$ into V and $V^\perp$. The idea of the Newton-Picard method is that we apply Newton's method to the part of the problem in V and a Picard iteration in the part of the problem in $V^\perp$.

We will now derive the Newton-Picard method. When we write V_p and V_q we think of them as being in the form of a $n_{ODE} \times p$ resp. $n_{ODE} \times (n_{ODE} - p)$ matrix, where p is the size of the basis, and where the columns of the matrices are the vectors in the bases. Thinking of V_p and V_q this way, we define the projectors:

$$P := V_p V_p^T$$

$$Q := V_q V_q^T = I - V_p V_p^T$$

Note that P projects onto V and Q projects onto $V^\perp$. Note that V_q is used purely for theoretical purposes and will not be explicitly computed, since it is quite big.

Any $y \in \mathbb{R}^n_{ODE}$ can be decomposed as $x = p + q$ with $p = Py$ and $Q = Qy$. If we decompose all vectors in the phase space of the ODE as $x = p + q$ in the left hand side of equation 3.4, we get

$$\begin{bmatrix} M - I & b_T & b_\gamma \\ c_s^T & d_{s,T} & d_{s,\gamma} \\ c_n^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} P\Delta y + Q\Delta y \\ \Delta T \\ \Delta \gamma \end{bmatrix} = - \begin{bmatrix} r(y,T,\gamma) \\ s(y,T,\gamma) \\ n(y,T,\gamma) \end{bmatrix} \tag{3.6}$$

Since the subspaces V and $V^\perp$ are orthogonal, we can separate the equations in the phase space $\mathbb{R}^n_{ODE}$ into equations in V and equations in $V^\perp$, to obtain the $2n_{ODE} + 2$ dimensional system:

$$\begin{bmatrix} M - I & M - I & b_T & b_\gamma \\ c_s^T & c_s^T & d_{s,T} & d_{s,\gamma} \\ c_n^T & c_n^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} Q\Delta y \\ P\Delta y \\ \Delta T \\ \Delta \gamma \end{bmatrix} = - \begin{bmatrix} r(y,T,\gamma) \\ s(y,T,\gamma) \\ n(y,T,\gamma) \end{bmatrix} \tag{3.7}$$

We substitute $V_q V_q^T$ for Q, and $V_p V_p^T$ for P:

$$\begin{bmatrix} M - I & M - I & b_T & b_\gamma \\ c_s^T & c_s^T & d_{s,T} & d_{s,\gamma} \\ c_n^T & c_n^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} V_q V_q^T \Delta y \\ V_p V_p^T \Delta y \\ \Delta T \\ \Delta \gamma \end{bmatrix} = - \begin{bmatrix} r(y,T,\gamma) \\ s(y,T,\gamma) \\ n(y,T,\gamma) \end{bmatrix} \tag{3.8}$$

By moving the factors V_q and V_p into the matrix, the system is once again n_{ODE} dimensional:

$$\begin{bmatrix} (M - I)V_q & (M - I)V_p & b_T & b_\gamma \\ c_s^T V_q & c_s^T V_p & d_{s,T} & d_{s,\gamma} \\ c_n^T V_q & c_n^T V_p & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} V_q^T \Delta y \\ V_p^T \Delta y \\ \Delta T \\ \Delta \gamma \end{bmatrix} = - \begin{bmatrix} r(y,T,\gamma) \\ s(y,T,\gamma) \\ n(y,T,\gamma) \end{bmatrix} \tag{3.9}$$

Now we multiply the first n_{ODE} rows of both sides of the equation by $[V_q\ V_p]^T$ on the left side.

$$\begin{bmatrix} V_q^T(M-I)V_q & V_q^T(M-I)V_p & V_q^T b_T & V_q^T b_\gamma \\ V_p^T(M-I)V_q & V_p^T(M-I)V_p & V_p^T b_T & V_p^T b_\gamma \\ c_s^T V_q & c_s^T V_p & d_{s,T} & d_{s,\gamma} \\ c_n^T V_q & c_n^T V_p & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} V_q^T \Delta y \\ V_p^T \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} V_q^T r \\ V_p^T r \\ s(y,T,\gamma) \\ n(y,t,\gamma) \end{bmatrix} \tag{3.10}$$

Since V is an invariant subspace of M, the vectors MV_p span V. Therefore, $V_q^T(M-I)V_p$ is zero at convergence. Thus, we neglect this term.

Similarly, $V_q^T b_T$ is zero at convergence as well. We have $b_T = \partial\phi(y,T,\gamma)/\partial T = f(\phi(y,T,\gamma),\gamma)$. Since $f(\phi(y,T,\gamma))$ is the eigenvector that corresponds to the eigenvalue one of the monodromy matrix M, and this eigenvector is in the subspace V, we have $V_q^T b_T = 0$ at convergence. Thus, we neglect $V_q^T b_T$, as well. [1] Hence, the system we want to solve is now:

$$\begin{bmatrix} V_q^T(M-I)V_q & 0 & 0 & V_q^T b_\gamma \\ V_p^T(M-I)V_q & V_p^T(M-I)V_p & V_p^T b_T & V_p^T b_\gamma \\ c_s^T V_q & c_s^T V_p & d_{s,T} & d_{s,\gamma} \\ c_n^T V_q & c_n^T V_p & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} V_q^T \Delta y \\ V_p^T \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} V_q^T r \\ V_p^T r \\ s(y,T,\gamma) \\ n(y,t,\gamma) \end{bmatrix} \tag{3.11}$$

The systems

$$\left[V_q^T(M-I)V_q\right]\left[V_q^T\Delta q_r\right] = -\left[V_q^T r\right] \tag{3.12}$$

$$\left[V_q^T(M-I)V_q\right]\left[V_q^T\Delta q_\gamma\right] = -\left[V_q^T b_\gamma\right] \tag{3.13}$$

can be solved by an iterative method, which we will describe in section 3.3. These systems are called the Q-systems. For now, we consider the Q-systems solved, and we rewrite the first block row of equation (3.11) in terms of the solutions $V_q^T\Delta q_r$, and $V_q^T\Delta q_\gamma$.

$$\begin{bmatrix} I_{n_{ODE}-p} & 0 & 0 & -V_q^T\Delta q_\gamma \\ V_p^T(M-I)V_q & V_p^T(M-I)V_p & V_p^T b_T & V_p^T b_\gamma \\ c_s^T V_q & c_s^T V_p & d_{s,T} & d_{s,\gamma} \\ c_n^T V_q & c_n^T V_p & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} V_q^T \Delta y \\ V_p^T \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} -V_q^T\Delta q_r \\ V_p^T r \\ s(y,T,\gamma) \\ n(y,t,\gamma) \end{bmatrix} \tag{3.14}$$

Note that since V_q and V_p are orthogonal have $V_p^T V_q = 0$, thus we have $V_p^T(M-I)V_q = V_p^T M V_q - V_p^T V_q = V_p M V_q$. Thus we have:

$$\begin{bmatrix} I_{n_{ODE}-p} & 0 & 0 & -V_q^T\Delta q_\gamma \\ V_p^T M V_q & V_p^T(M-I)V_p & V_p^T b_T & V_p^T b_\gamma \\ c_s^T V_q & c_s^T V_p & d_{s,T} & d_{s,\gamma} \\ c_n^T V_q & c_n^T V_p & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} V_q^T \Delta y \\ V_p^T \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} -V_q^T\Delta q_r \\ V_p^T r \\ s(y,T,\gamma) \\ n(y,t,\gamma) \end{bmatrix} \tag{3.15}$$

Note that the equations in the upper $n_{ODE}-p$ rows can be written as:

$$V_q^T\Delta y - V_q^T\Delta q_\gamma \Delta\gamma = V_q^T\Delta q_r \tag{3.16}$$

[1] Even though we choose to neglect $V_q^T b_T$, is it not necessary. In [19], K. Lust mentions that $V_q^T b_T$ can be treated in a similar manner as $V_q^T b_\gamma$

In each of the three lower block rows, we substitute the term with $V_q^T \Delta y$ with $V_q^T \Delta q_\gamma \Delta\gamma + V_q^T \Delta q_r$. In other words, we use Gaussian elimination to introduce zero's in the lower three positions of the first block column of the matrix. This is relatively simple, because of the zeros in the first block row.

$$\begin{bmatrix} I_{N-p} & 0 & 0 & -V_q^T \Delta q_\gamma \\ 0 & V_p^T(M-I)V_p & V_p^T b_T & V_p^T b_\gamma + V_p^T M V_q V_q^T \Delta q_\gamma \\ 0 & c_s^T V_p & d_{s,T} & d_{s,\gamma} + c_s^T V_q V_q^T \Delta q_\gamma \\ 0 & c_n^T V_p & d_{n,T} & d_{n,\gamma} + c_n^T V_q V_q^T \Delta q_\gamma \end{bmatrix} \begin{bmatrix} V_q^T \Delta y \\ V_p^T \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} -V_q^T \Delta q_r \\ V_p^T r + V_p^T M V_q V_q^T \Delta q_r \\ s(y,T,\gamma) + c_s^T V_q V_q^T \Delta q_r \\ n(y,t,\gamma) + c_n^T V_q V_q^T \Delta q_r \end{bmatrix} \quad (3.17)$$

We can now solve the the system

$$\begin{bmatrix} V_p^T(M-I)V_p & V_p^T b_T & V_p^T b_\gamma + V_p^T M V_q V_q^T \Delta q_\gamma \\ c_s^T V_p & d_{s,T} & d_{s,\gamma} + c_s^T V_q V_q^T \Delta q_\gamma \\ c_n^T V_p & d_{n,T} & d_{n,\gamma} + c_n^T V_q V_q^T \Delta q_\gamma \end{bmatrix} \begin{bmatrix} V_p^T \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} V_p^T r + V_p^T M V_q V_q^T \Delta q_r \\ s(y,T,\gamma) + c_s^T V_q V_q^T \Delta q_r \\ n(y,T,\gamma) + c_n^T V_q V_q^T \Delta q_r \end{bmatrix} \quad (3.18)$$

Which is considerably smaller than (3.4).

When solving the Q-systems, we actually solve the simpler n_{ODE}-dimensional systems (3.19), instead of the $n_{ODE} - p$-dimensional systems (3.12) and (3.13).

$$\left[M - I\right]\left[\Delta q_r\right] = -\left[Qr\right] \qquad \left[M - I\right]\left[\Delta q_\gamma\right] = -\left[Qb_\gamma\right] \quad (3.19)$$

subject to $\Delta q_\gamma, \Delta q_r \in V^\perp$, and where we only require (near) equality in the equations (3.19) in $V^\perp$. Hence, we will find Δq_r and Δq_γ such that:

$$V_q V_q^T \Delta q_r = Q \Delta q_r = \Delta q_r$$

$$V_q V_q^T \Delta q_\gamma = Q \Delta q_\gamma = \Delta q_\gamma$$

and thus equation (3.18) becomes:

$$\begin{bmatrix} V_p^T(M-I)V_p & V_p^T b_T & V_p^T b_\gamma + V_p^T M \Delta q_\gamma \\ c_s^T V_p & d_{s,T} & d_{s,\gamma} + c_s^T \Delta q_\gamma \\ c_n^T V_p & d_{n,T} & d_{n,\gamma} + c_n^T \Delta q_\gamma \end{bmatrix} \begin{bmatrix} V_p^T \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} V_p^T r + V_p^T M \Delta q_r \\ s(y,T,\gamma) + c_s^T \Delta q_r \\ n(y,T,\gamma) + c_n^T \Delta q_r \end{bmatrix} \quad (3.20)$$

3.3 Solving the Q-systems

The n_{ODE}-dimensional vectors Δq_r and Δq_γ are computed using the following iterative method. Kurt Lust also described how to use $GMRES$ to compute Δq_r and Δq_γ, but that

is outside the scope of this thesis. Moreover, it is unclear if $GMRES$ would perform better.

Input:

- a basis V_p for an invariant subspace V of M associated to the dominant eigenvalues (in norm) of M. In this algorithm basis V_p is used as a matrix whose columns are the basis vectors.
- a right hand side rhs of the Q-system $(M - I)\Delta q = -rhs$
- a way to compute Mx given a vector x. We denote the function call to compute Mx from x by $compute\_M(x)$. Note that we make a distinction between the stored values $M\Delta q$, and the function call $compute\_M(\Delta q)$ to compute it.
- A maximum number of iterations $max\_iterations$. If, for instance, the stepsize in the continuation is too large, the solution might not converge. For such a contingency, we stop after a certain number of iterations, so that the stepsize can be reduced.

Result:

- $V_q\Delta q$ and $MV_q\Delta q$

$residual \leftarrow rhs - V_pV_p^T rhs$
$M\Delta q = 0$
if $\|residual\| < tolerance$ **then**
  $\Delta_q = 0$
  return $M\Delta q$ and Δq
end
for $iteration\ number = 1$ ***up to*** $max\_iterations$ **do**
  $\Delta q \leftarrow M\Delta q + rhs$
  $\Delta q \leftarrow \Delta q - V_pV_p^T\Delta q$
  $M\Delta q \leftarrow compute\_M(\Delta q)$
  $residual \leftarrow rhs + M\Delta q - \Delta q$
  $residual \leftarrow residual - V_pV_p^T residual$
  if $\|residual\| < tolerance$ **then**
    return $M\Delta q$ and Δq
  end
end

Algorithm 3: How to solve the Q-system

3.4 Continuation of the subspaces

In this section, we will explain how to compute the basis V_p, and how to update it each continuation step. To compute V_p, the function `eigs` in Matlab was used. The function `eigs` can accept a function that applies a linear operator to a vector. By supplying the Monodromy operator of the current approximation of the cycle to `eigs`, we find the eigenvectors of associated to the eigenvalues of greatest norm. Denote the monodromy operator by M. The result Mx can be computed using Algorithm 4. It is implemented in the file `Continuer/+NewtonPicard/+SingleShooting/monodromy_map.m` in the CL_MATCONTL source. This function is also the implementation of the function compute_M in algorithm 3.

When computing the current approximation of the cycle and evaluating the mon-

odromy map, one can specify tolerances to the time integration function. The time integration function used is the Matlab ODE solver `ode15s` (although one can easily specify in the CL_MATCONTL commands that another Matlab or Matlab compatible ODE solver should be used). Choosing the tolerance of the ODE solver is a matter of balancing accuracy versus computation time. Choosing low tolerances will increase accuracy. Low tolerances will also enable bigger step sizes in the continuation. On the other hand, the computation time per step will increase. Choosing extremely low tolerances will guarantee that the continuation will proceed with a minimum of failed steps, but will likely cause the computation time per step to be unnecessarily high. On the other hand, one should set absolute integration tolerance lower than the continuation tolerance, otherwise the continuation steps will probably not converge down to the continuation tolerance.

Input:

- a vector x for which we want to compute Mx
- the current approximation of the period T of the limit cycle.
- the current approximation y_c of the periodic orbit

Output:

- Mx

1. Solve the initial value problem $y(0) = x, \frac{dy}{dt} = f_y(y_c(t))y$
   By default, the Matlab function `ode15s` is used for this.
2. return $y(T)$

Algorithm 4: How to compute Mx

If a basis V_p from a previous continuation step is already known, this basis can be continued. The advantage of this is that the changes in the basis will be smooth. That is, out of all the possible orthonormal bases that span the subspace, with continuation we find a basis that is close to the basis in the previous step. Continuation of a subspace of a monodromy matrix is done as shown in algorithm 5, and is implemented in the file `Continuer/+NewtonPicard/+SingleShooting/continue_subspace.m` This algorithm is based on algorithm 1 in [18].

Input:

- the basis V_p used for the previous Newton-Picard correction

Result:

- a basis V_p to be used for the current Newton-Picard correction

Add random some vectors to the basis $V_p = \{v_i\}_i$
effective_ basis_ size $= 0$
for *iteration* $= 1$ *up to max_ iteration* **do**
  if *iteration* > 1 **then**
    for $i =$ *effective_ basis_ size + 1 up to the size of the basis* **do**
      $v_i \leftarrow w_i$
    end
    orthonormalize the column vectors of V_p starting at
    column *effective_ basis_ size* $+ 1$
  end
  for $i =$ *effective_ basis_ size up to the size of the basis* **do**
    $w_i \leftarrow M v_i$
  end
  $W \leftarrow \{w_i\}_i$
  $U \leftarrow V_p W$
  compute the Schur factorization Y, S of U (Y, S such that $UY = YS$, where S is upper triangular, and Y is unitary) such that the Schur vectors are ordered according to the norm of the associated eigenvalues.
  $V_p \leftarrow V_p Y$
  for $k =$ *size of the basis down to 1* **do**
    Compute the largest singular value s of the first k columns of $W - V_p S$
    if $s <$*tolerance* **then**
      effective_ basis_ size $\leftarrow k$
      break
    end
  end
  if *effective_ basis_ size* $>=$ *required_ basis_ size* **then**
    return V_p
  end
end

Algorithm 5: Continuation of the basis

3.5 Summary

To summarize, combining single shooting with Newton-Picard entails that we replace the Newton correction (algorithm 2) with a more sophisticated variant summarized in algorithm 6. The while loop (without the statements inside it) in algorithm 6 is implemented in the file `Continuer/+NewtonPicard/do_corrections.m` in the CL_MATCONTL source, and the statements inside the while loop are implemented in the file `Continuer/+NewtonPicard/+SingleShooting/do_one_correction.m`.

The number of evaluations of the monodromy map (Algorithm 4) needed for one correction cannot be determined exactly a priori. We do know that MV_p must be computed, which requires one evaluation of the Monodromy map for every vector in V_p. Hence, computing MV_p requires p evaluations of the monodromy map. The number of evaluations of the monodromy map needed for solving of the Q-systems cannot be determined a priori. In rare cases it may not need any evaluations at all.

done $\leftarrow$ false
while *not done* **do**

1. Compute the current approximation y_c of the cycle by integration from the starting point y, by time integration using `ode15s`, or a similar time integration method.

2. Compute the subspace V_p using the `eigs` function in Matlab or continue the subspace V_p using algorithm 5. When calling either of these functions, pass algorithm 4 and an argument, and pass the result y_c computed in step 1 to algorithm 4.

3. Compute Δq_r, $M\Delta q_r$, Δq_γ, and $M\Delta q_\gamma$ using algorithm 3, again passing algorithm 4 as an argument, and passing y_c to algorithm 4.

4. Compute the matrix J:

$$J = \begin{bmatrix} V_p^T(M-I)V_p & V_p^T b_T & V_p^T b_\gamma + V_p^T M\Delta q_\gamma \\ c_s^T V_p & d_{s,T} & d_{s,\gamma} + c_s^T \Delta q_\gamma \\ c_n^T V_p & d_{n,T} & d_{n,\gamma} + c_n^T \Delta q_\gamma \end{bmatrix}$$

$$= \begin{bmatrix} V_p^T M V_p - I & V_p^T f(\phi(T), p) & V_p^T \left(\frac{\partial(\phi(T))}{\partial\gamma} + M\Delta q_\gamma \right) \\ f(y_0^0, \gamma_0)^T V_p & 0 & f(y_0^0, \gamma_0)^T \Delta q_\gamma \\ v_y^T V_p & v_T & v_\gamma + v_y^T \Delta q_\gamma \end{bmatrix}$$

with:
y_0^0 equal to the coordinates of the point on the cycle at the previous continuation point (see equation (3.2))
γ_0 equal to the parameter of the system ODEs at the previous continuation point.
and v_y, v_T, and v_γ equal to the parts of the continuation tangent vector associated to the point on the cycle x, the period T and the parameter γ, respectively.

5. Solve the linear system: $J \begin{bmatrix} V_p^T \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} V_p^T r + V_p^T M\Delta q_r \\ s(y,T,\gamma) + c_s^T \Delta q_r \\ n(y,T,\gamma) + c_n^T \Delta q_r \end{bmatrix}$

6. Compute the corrected values of y, T, and γ:

$$\begin{aligned} \Delta y &\leftarrow V_p V_p^T \Delta y + \Delta q_r \\ y &\leftarrow y + \Delta y \\ T &\leftarrow T + \Delta T \\ \gamma &\leftarrow \gamma + \Delta\gamma \end{aligned}$$

7. we are done if:

$$\left\| \left(\left\| r(y,T,\gamma) \right\|_\infty, s(y,T,\gamma), n(y,T,\gamma) \right) \right\|_\infty < tolerance_F$$

and

$$\left\| (\Delta y, \Delta T, \Delta\gamma) \right\|_\infty < tolerance_{\Delta x}$$

end
Algorithm 6: how to compute a Newton-Picard corrections for single shooting

Chapter 4

Continuation of cycles using multiple shooting with Newton-Picard

In his PhD Thesis Kurt Lust has shown that the methods described in chapter 3 can be extended to multiple shooting. However, no implementation of any multiple shooting method for large systems, which is publicly available, has been found, though others have also been working on multiple shooting methods for cycles in large systems [20]. In this chapter we will discuss Lust's methods, and briefly comment on how they were implemented in Matlab.

At each step in a continuation of periodic orbits using multiple shooting the following equations have to be solved for the set of vector valued variables $\{x_i \mid 1 \leq i \leq m\}$:

$$\begin{cases} \phi(x_1, \Delta s_1 T, \gamma) - x_2 & = 0 \\ \phi(x_2, \Delta s_2 T, \gamma) - x_3 & = 0 \\ \phi(x_2, \Delta s_3 T, \gamma) - x_3 & = 0 \\ .... & \\ \phi(x_{m-1}, \Delta s_{m-1} T, \gamma) - x_m & = 0 \\ \phi(x_m, \Delta s_m T, \gamma) - x_1 & = 0 \\ s(x_0, ..., x_m, T, \gamma) & = 0 \\ n(x_0, ..., x_m, T, \gamma) & = 0 \end{cases} \tag{4.1}$$

where $\phi(x_0, T, \gamma)$ is the solution of the initial value problem $dx/dt = f(x, \gamma)$, $x(0) = x_0$ evaluated at T, and where the given function f defines the system of ODEs. Here we use the notation of [19, K. Lust, PhD Thesis], except for the indices of Δx, where we use Δx_1 up to Δx_m, instead of Δx_0 up to Δx_{m-1}, to be consistent with the 1-based indexation convention of Matlab. To solve these nonlinear equation's one can apply Newton's method. This leads to the linear system:

$$\begin{bmatrix} G_1 & -I & & & b_{T,1} & b_{\gamma,1} \\ & G_2 & -I & & b_{T,2} & b_{\gamma,2} \\ & & \ddots & \ddots & \dots & \dots \\ -I & & & G_m & b_{T,m} & b_{\gamma,m} \\ c_{s,1}^T & c_{s,2}^T & \dots & c_{s,m}^T & d_{s,T} & d_{s,\gamma} \\ c_{n,1}^T & c_{n,2}^T & \dots & c_{n,m}^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_m \\ \Delta T \\ \Delta \gamma \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \\ s \\ n \end{bmatrix} \tag{4.2}$$

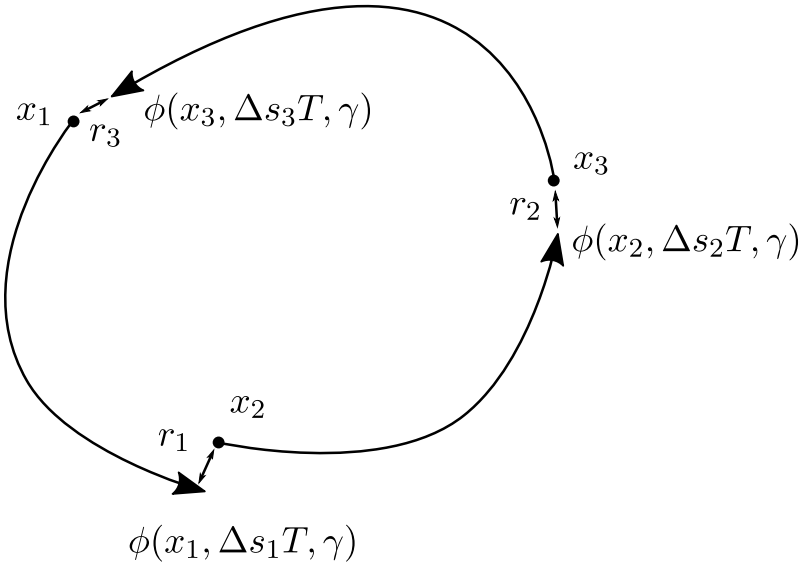


Figure 4.1: Diagram of multiple shooting

where:

$$r_i(x_{i-1}, x_i, T, \gamma) = \begin{cases} \phi(x_i, \Delta s_i T, \gamma) - x_{i+1} & \text{if} \quad i < m \\ \phi(x_m, \Delta s_i T, \gamma) - x_1 & \text{if} \quad i = m \end{cases} \tag{4.3}$$

and:

$$\begin{aligned} G_i &= \frac{\partial \phi(x_i, \Delta s_i T, \gamma)}{\partial x_i} \\ B_i = \begin{bmatrix} b_{T,i} & b_{\gamma,i} \end{bmatrix} &= \frac{\partial r_i}{\partial (T, \gamma)} \\ C_i = \begin{bmatrix} c_{s,i}^T \\ c_{n,i}^T \end{bmatrix} &= \frac{\partial (s, n)}{\partial x_i} \\ D = \begin{bmatrix} d_{s,T} & d_{s,\gamma} \\ d_{n,T} & d_{n,\gamma} \end{bmatrix} &= \frac{\partial (s, n)}{\partial (T, \gamma)} \end{aligned} \tag{4.4}$$

Let $M_1 = G_m G_{m-1} \dots G_1$. Note that M_1 is the monodromy matrix of the cycle relative to x_1, if the continuation step has converged. Define also $M_i = G_{i-1} G_{i-2} \dots G_1 G_m \dots G_{i+1} G_i$, i.e. M_i is the product G's that ends with G_i. Note that this choice of indexation for $M_i, P_i, Q_i, V_{p,i}, V_{q,i}$ is another deviation from the notations in Kurt Lust' PhD thesis [19]. Note that M_i is the monodromy matrix of the cycle relative to x_i, if the continuation step has converged.

The multiple shooting method for large systems can be derived in a manner similar to the derivation of the single shooting method derived in chapter 3. We will not go into as much detail as in chapter 3. We introduce the projectors $P_i = V_{p,i} V_{p,i}^T$ and $Q_i = I - P_i = V_{q,i} V_{q,i}^T$, where $V_{p,i}$ is a basis for the subspace V_i of M_i corresponding to the set of leading eigenvalues in norm. When we write $V_{p,i}$ we think of it as being in the form of a $n_{ODE} \times p$ matrix, where p is the size of the basis, and where the columns are the vectors in the basis. Note that at convergence the eigenvalues of every M_i are the same. In the linearized system (4.2) we split Δx_i into $P_i \Delta x_i + Q_i \Delta x_i$. We split r_i into $P_{i+1} r_i + Q_{i+1} r_i$ for $i < m$ and r_m into $P_1 r_m + Q_1 r_m$, since r_i is the residual near the point x_{i+1}, and r_m is the residual near

x_1. See also figure 4.1.

$$\begin{bmatrix} G_1 & -I & & & b_{T,1} & b_{\gamma,1} \\ & G_2 & -I & & b_{T,2} & b_{\gamma,2} \\ & & \ddots & \ddots & \dots & \dots \\ -I & & & G_m & b_{T,m} & b_{\gamma,m} \\ c_{s,1}^T & c_{s,2}^T & \dots & c_{s,m}^T & d_{s,T} & d_{s,\gamma} \\ c_{n,1}^T & c_{n,2}^T & \dots & c_{n,m}^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} P_1\Delta x_1 + Q_1\Delta x_1 \\ P_2\Delta x_2 + Q_2\Delta x_2 \\ \vdots \\ P_m\Delta x_m + Q_m\Delta x_m \\ \Delta T \\ \Delta\gamma \end{bmatrix} = \begin{bmatrix} P_2r_1 + Q_2r_1 \\ P_3r_2 + Q_3r_2 \\ \vdots \\ P_mr_{m-1} + Q_mr_{m-1} \\ P_1r_m \\ s \\ n \end{bmatrix} \tag{4.5}$$

In a manner similar to the single shooting case, two systems are derived from this. One system is Kurt Lust calls the P-system and the other the Q-system. The Q-system is solved by an iterative method, and the P system is solved directly. The P-system is:

$$\begin{bmatrix} F_{pp}^0 & \{V_{p,i}^T(B_i + \Delta q_{i,\gamma})\} \\ \{C_i^T V_{p,i}\} & D + \sum_{i=1}^m C_i^T \Delta q_{i,\gamma} \end{bmatrix} \begin{bmatrix} V_{p,1}^T\Delta x_1 \\ V_{p,2}^T\Delta x_2 \\ \vdots \\ V_{p,m}^T\Delta x_m \\ \Delta T \\ \Delta\gamma \end{bmatrix} = \begin{bmatrix} V_{p,2}^T r_1 \\ V_{p,3}^T r_2 \\ \vdots \\ V_{p,1}^T r_m \\ s \\ n \end{bmatrix} \tag{4.6}$$

where:

$$F_{pp}^0 = \begin{bmatrix} V_{p,2}^T G_1 V_{p,1} & -I & & & \\ & V_{p,3}^T G_1 V_{p,2} & -I & & \\ & & \ddots & \ddots & \\ & & & V_{p,m}^T G_{m-1} V_{p,m-1} & -I \\ -I & & & & V_{p,1}^T G_m V_{p,m} \end{bmatrix} \tag{4.7}$$

$$\{V_{p,i}^T(B_i + \Delta q_{i,\gamma})\} = \begin{bmatrix} V_{p,2}^T b_{T,1} & V_{p,2}(b_{\gamma,1} + \Delta q_{1,\gamma}) \\ V_{p,3}^T b_{T,2} & V_{p,3}(b_{\gamma,2} + \Delta q_{2,\gamma}) \\ \vdots & \vdots \\ V_{p,m}^T b_{T,m-1} & V_{p,m}(b_{\gamma,m-1} + \Delta q_{m-1,\gamma}) \\ V_{p,1}^T b_{T,m} & V_{p,1}(b_{\gamma,m} + \Delta q_{m,\gamma}) \end{bmatrix} \tag{4.8}$$

and:

$$\{C_i^T V_{p,i}\} = \begin{bmatrix} c_{s,1}^T V_{p,1} & c_{s,2}^T V_{p,2} & \dots & c_{s,m}^T V_{p,m} \\ c_{n,1}^T V_{p,1} & c_{n,2}^T V_{p,2} & \dots & c_{n,m}^T V_{p,m} \end{bmatrix} \tag{4.9}$$

The sequences of m n_{ODE}-dimensional vectors $\{\Delta q_{i,\gamma}\}_{i=1}^m$ and $\{\Delta q_{i,r}\}_{i=1}^m$ are the solutions of the following Q-systems:

$$G^0 \begin{bmatrix} \Delta q_{1,r} \\ \vdots \\ \Delta q_{m,r} \end{bmatrix} = \begin{bmatrix} Q_2 r_1 \\ \vdots \\ Q_m r_{m-1} \\ Q_1 r_m \end{bmatrix} \quad \text{and} \quad G^0 \begin{bmatrix} \Delta q_{1,\gamma} \\ \vdots \\ \Delta q_{m,\gamma} \end{bmatrix} = \begin{bmatrix} Q_2 b_{\gamma,1} \\ \vdots \\ Q_m b_{\gamma,m-1} \\ Q_1 b_{\gamma,1} \end{bmatrix}$$

with:

$$G^0 = \begin{bmatrix} G_1 & -I & & & \\ & G_2 & -I & & \\ & & \ddots & \ddots & \\ & & & G_{m-1} & -I \\ -I & & & & G_m \end{bmatrix} \tag{4.10}$$

4.1 Solving the Q-systems

The Q-systems are solved using algorithm 7, which is based on algorithm 6.2 in [19]. It is implemented in the file
`Continuer/+NewtonPicard/+MultipleShooting/solve_Q_system.m`

Input:

- bases $V_{p,i}$ for the subspaces V_i of M_i
- a right hand side rhs of the Q-system
- a way to compute $G_i x$ given a vector x. We denote the function call to compute $G_i x$ from x by $compute\_G(x, i)$. Note that we make a distinction between the stored values $G\Delta q_i$, and the function call $compute\_G(\Delta q_i, i)$ to compute it.
- A maximum number of iterations $max\_iterations$. If, for instance, the stepsize in the continuation is too large, the solution might not converge. For such a contingency, we stop after a certain number of iterations, so that the stepsize can be reduced.

Result:

- Δq_i and $G_i \Delta q_i$ for $1 < i < m$.

for $i = 1$ *up to* m **do**
  $i_{next} \leftarrow \begin{cases} i+1 & \text{if } i < m \\ 1 & \text{if } i = m \end{cases}$
  /* We project rhs_i on the subspace spanned by $V_{q,i_{next}}$. This is equivalent to $rhs_i \leftarrow Q_{i_{next}} rhs_i$, but since we dont't want to compute Q_i, we do: */
  $rhs_i \leftarrow rhs_i - V_{p,i_{next}} V_{p,i_{next}}^T rhs_i$
end
$\Delta q_i = 0$ for all i
$G\Delta q_i = 0$ for all i
for *iteration number* $= 1$ *up to* $max\_iterations$ **do**
  for $i = 2$ *up to* m **do**
    $\Delta q_i \leftarrow G\Delta q_{i-1} + rhs_{i-1}$
    $\Delta q_i \leftarrow \Delta q_i - V_i V_i^T \Delta q_i$
    $G\Delta q_i \leftarrow compute\_G(\Delta q_i, i)$
  end
  $condensed\_residual \leftarrow G\Delta q_m + rhs_m$
  $condensed\_residual \leftarrow condensed\_residual - V_1 V_1^T condensed\_residual$
  if $\|condensed\_residual\| < tolerance$ **then**
    return Δ_q and $G\Delta_q$
  end
  else
    $G\Delta q_1 \leftarrow compute\_G(\Delta q, 1)$
  end
end

Algorithm 7: how to solve the Q-systems for multiple shooting

4.2 Computation of the subspaces

We now discuss how to compute V_i ($1 \leq i \leq m$). Kurt Lust provides a sophisticated algorithm in his PhD thesis for the continuation of the bases. However, the implementation of this algorithm was outside the scope of this thesis. Instead, we use the same methods used for single shooting to continue V_1, and then compute V_i for $2 \leq i \leq m$ by applying G_{i-1} to each vector of V_{i-1}, and the orthonormalizing the resulting set of vectors.

4.3 Adaptation of the time mesh

In multiple shooting we continue the cycle by continuing multiple points on the cycle. There is some freedom in choosing the time intervals between the points. We wish to choose the time intervals in such a way that it increases the domain of attraction of the Newton corrections. The way we do this is to distribute the points along the cycle such that the norm of the gradient along the cycle between each mesh point is the same.

4.4 Summary

We summarize the algorithm discussed in this chapter in algorithm 8. The while loop, without the statements inside it in implemented in the file

`Continuer/+NewtonPicard/do_corrections.m`

in the CL_MATCONTL source. The statements inside the while loop are implemented in the file

`Continuer/+NewtonPicard/+MultipleShooting/do_one_correction.m`

in the CL_MATCONTL source.

done $\leftarrow$ false
while *not done* **do**

- Compute the current approximation of the cycle by integration from the starting point x_1
- Compute the the partial cycle trajectories form starting point x_i for a time interval of length $\Delta s_i T$ for $2 \leq i \leq m$
- Compute the subspace V_1 using the `eigs` function in Matlab or continue the subspace V_p using algorithm 5.
- Compute the subspaces V_i for $2 \leq i \leq m$ by applying G_{i-1} to the vectors of V_{i-1} and orthonormalize the result.
- Compute $\Delta q_{i,r}$, $G_i \Delta q_r$, $\Delta q_{i,\gamma}$, and $G_i \Delta q_{i,\gamma}$ for $1 \leq i \leq m$ using algorithm 7.
- Compute the matrix J:

$$\begin{bmatrix} F_{pp}^0 & \{V_{p,i}^T (B_i + \Delta q_{i,\gamma})\} \\ \{C_i^T V_{p,i}\} & D + \sum_{i=1}^m C_i^T \Delta q_{i,\gamma} \end{bmatrix}$$

  For the definitions of F_{pp}^0, $\{V_{p,i}^T (B_i + \Delta q_{i,\gamma})\}$, and $\{C_i^T V_{p,i}\}$, see equations (4.7), (4.8), and (4.9).

- Solve the linear system:

$$J \begin{bmatrix} V_{p,1}^T \Delta x_1 \\ V_{p,2}^T \Delta x_2 \\ \vdots \\ V_{p,m}^T \Delta x_m \\ \Delta T \\ \Delta \gamma \end{bmatrix} = \begin{bmatrix} V_{p,2}^T r_1 \\ V_{p,3}^T r_2 \\ \vdots \\ V_{p,1}^T r_m \\ s \\ n \end{bmatrix}$$

- Compute the corrected values of x_i $(1 \leq i \leq m)$, T, and γ:

$$\begin{aligned} \Delta x_i &\leftarrow V_p V_p^T \Delta x_i + \Delta q_{i,r} && (1 \leq i \leq m) \\ x_i &\leftarrow x_i + \Delta x_i && (1 \leq i \leq m) \\ T &\leftarrow T + \Delta T \\ \gamma &\leftarrow \gamma + \Delta \gamma \end{aligned}$$

- we are done if:

$$\left\| (\left\| r_1 \right\|_\infty, \ldots, \left\| r_m \right\|_\infty, s, n,) \right\|_\infty < tolerance_F$$

and

$$\left\| (\left\| \Delta x_1 \right\|_\infty, \ldots, \left\| \Delta x_m \right\|_\infty, \Delta T, \Delta \gamma) \right\|_\infty < tolerance_{\Delta x}$$

end

Algorithm 8: how to compute a Newton-Picard corrections for multiple shooting

Chapter 5

Detection of Bifurcations

When continuing a branch of cycles, various bifurcations can occur. These are points in the branch where the stability of the cycle changes. There are three types of codimension one bifurcations. These types are period doubling points (PD), a limit points of cycles (LPC), and Neimark-Sacker bifurcations (NS). Although a branching point of cycles (BPC) is a codimension 2 phenomenon, one can encounter them in branches of cycles due to symmetries in the system.

We wish to detect and locate these bifurcations. The methods that were implemented in Matcont were not always adequate for large systems.

5.1 Detection of Limit Points of Cycles (LPC)

The only bifurcation detection method that could be used without any changes was the detection method for limit points of cycles. LPCs are detected by monitoring for a sign change in the coordinate v_γ of the continuation tangent vector that corresponds to the active parameter. The active parameter is the parameter of the system of ODEs that is being changed. If the sign of v_γ changes, an LPC has been found. Due to the simplicity of this method it can be used for large systems as well.

5.2 Detection of Period Doubling points (PD)

The method of detecting PDs was changed. Let M be the monodromy matrix of a cycle. In Matcont PDs were detected by computing the determinant of $M + I$ and monitoring this result for a sign change. This works, since at a PD one of the multipliers of the cycles is equal to -1. The multipliers are the eigenvalues of the monodromy matrix. Therefore, $M + I$ has a zero eigenvalues at a PD. Thus, $\det(M + I)$ changes sign at a PD.

However, if we use Newton Picard, the monodromy matrix is not readily available. The whole point of Newton Picard is to avoid computing the monodromy matrix. Therefore, to detect PDs, we simply look at a subset of the Floquet multipliers. This subset S will be the subset of eigenvalues of which the norm is greater than a certain value ρ, which can be configured by the user. To detect bifurcations, ρ must less than one. This way, we compute only the multipliers that are relevant to detecting bifurcations. To detect PDs, we compute:

$$\psi_{PD} = \prod_{\mu \in S} (\mu + 1)$$

The function ψ_{PD} changes sign when a PD is passed, since then one of the multipliers

crosses the unit circle at -1, thereby changing the sign of one of the factors in the product. To accommodate for the changing size of S, we recompute ψ_{PD} every time a the size of S changes.

5.3 Detection of Neimark-Sacker bifurcations (NS)

When an NS bifurcation occurs, a pair of complex eigenvalues crosses the unit circle. To detect such a crossing, Matcont computed the bialternate product of the monodromy matrix. The eigenvalues of bialternate product B of the monodromy matrix are exactly the products of all pairs of eigenvalues of the monodromy matrix. Therefore, at a NS bifurcation, one of the eigenvalues of B is equal to one, since when a complex pair crosses the unit circle, it's product is equal to one.

However, as we said before, the monodromy matrix is not readily available. Therefore, as with period doubling points, we look at the subset of multipliers of which the norm is greater than a certain value ρ. We count the number of complex eigenvalues with norm greater than one. When this number changes, a Neimark-Sacker bifurcation has occurred. It could happen that a pair of real multipliers with norm greater than one becomes complex, but there are usually not so many multipliers with norm greater than one. Hence, this method will not produce many false positives.

5.4 Detection of Branching Points of Cycles (BPC)

At a branching point of cycles, two of the multipliers are equal to one. That is, in addition to the trivial multipliers that is always equal to one, at a BPC there is another multiplier equal to one. Hence, one way to detect a BPC is to compute

$$\psi_{BPC} = \prod_{\mu \in S^*} (\mu - 1),$$

and monitor for sign changes. Here, S^* is a subset of multipliers with norm greater than $\rho < 1$, and with the trivial multiplier removed. However, ψ_{BPC} also changes sign at a limit point of cycles, since at a limit point of cycles two of the multipliers are equal to one as well. Therefore, if ψ_{BPC} changes sign, we check if an LPC has been detected. The software will then indicate a BPC has been found, if no LPC has been detected.

One potential downside of this method of detecting BPCs is that if a BPC and an LPC occur between the same two continuation points, the BPCs will not be detected. Therefore, it might be worthwhile to detect BPC by detecting a rank drop in the Jacobian of the continuation. This way of detection BPCs was already implemented in Matcont for continuation of cycles by means of orthogonal collocation, but has not been successfully ported to CL_MATCONTL yet. Detecting BPCs by detecting a rank drop in the Jacobian matrix of the continuation could also work for single shooting and multiple shooting, but this was outside the scope of this thesis.

This problem of detecting BPCs in the presence of LPCs in especially problematic in case of a pitchfork bifurcation of cycles. In a pitchfork bifurcation a BPC and an LPC occur at the same point. In this case on the branch with the LPC no BPC will be detected. It is, however, still possible to detect the BPC on the other branch.

Another limitation of this method of detecting BPCs it may give false positives in certain systems, if the accuracy of the multipliers is low. In particular, if there are multipliers within the range of accuracy of the trivial multiplier, then one of these multipliers, instead

of the trivial multiplier is removed from the set of multipliers to form the set S^*, which may result is a sign change of ψ_{BPC} even if there is no BPC.

This problem was noticed when continuing the fusion system with the number of mesh points N equal to 75 using orthogonal collocation. When computing multipliers with the methods outlined in section 2.1.2, it was observed that the multipliers included many real multipliers just below one. When computing multipliers using the `eigs` function in Matlab, such values are not seen. Furthermore, there are less of these just-less-than-one-multipliers when the number of mesh interval is increased. Hence, these just-less-than-one multipliers are most likely due to numerical errors.

Hence, one can mostly avoid detecting false BPCs by either using a many mesh intervals or by applying the Matlab function `eigs` to action of the monodromy map. When using `eigs`, one evaluates the action of the monodromy map using algorithm 4.

Chapter 6

Examples

6.1 The Brusselator

The first example of a system with cycles is the Brusselator reaction-diffusion model. The model consists of system of two PDEs:

$$\begin{aligned} \frac{\partial X}{\partial t} &= \frac{D_X}{L^2}\frac{\partial^2 X}{\partial z^2} + X^2Y - (B+1)X + A \\ \frac{\partial X}{\partial t} &= \frac{D_Y}{L^2}\frac{\partial^2 Y}{\partial z^2} - X^2Y + BX \end{aligned} \tag{6.1}$$

with the Dirichlet boundary conditions:

$$\begin{aligned} X(t, z=0) = X(t, z=1) = A \\ Y(t, z=0) = Y(t, z=1) = B/A \end{aligned} \tag{6.2}$$

We use finite differences with an equidistant grid to discretize in space. For $i \in \{1, 2, ..., N\}$ let $X_i(t)$ be the approximation of $X(t, z = ih)$ and similarly let $Y_i(t)$ be the approximation of $Y(t, z = ih)$, where $h = 1/(N+1)$, and N is the number of grid points. Note that all grid points are in the interior of the interval $[0, 1]$, since the Dirichlet boundary conditions fix X and Y at the boundaries. To ease notation we define the reaction terms at grid point i as:

$$\begin{aligned} R_{X,i} &:= X_i^2 Y_i - (B+1)X_i + A \\ R_{Y,i} &:= -X_i^2 Y_i + BX \end{aligned} \tag{6.3}$$

where we drop the argument t of $R_{X,i}$, $R_{Y,i}$, X_i and Y_i for sake of legibility. For the two grid points h and $1-h$ that are near the boundary we have:

$$\begin{aligned} \frac{\partial^2 X}{\partial z^2}\Big|_{z=h} &\approx \frac{X(t,0) - 2X(t,h) + X(t,2h)}{h^2} &&\approx \frac{A - 2X_1 + X_2}{h^2} \\ \frac{\partial^2 Y}{\partial z^2}\Big|_{z=h} &\approx \frac{Y(t,0) - 2Y(t,h) + Y(t,2h)}{h^2} &&\approx \frac{B/A - 2Y_1 + Y_2}{h^2} \\ \frac{\partial^2 X}{\partial z^2}\Big|_{x=1-h} &\approx \frac{X(t,1-2h) - 2X(t,1-h) + X(t,1)}{h^2} &&\approx \frac{X_{N-1} - 2X_N + A}{h^2} \\ \frac{\partial^2 Y}{\partial z^2}\Big|_{x=1-h} &\approx \frac{Y(t,1-2h) - 2Y(t,1-h) + Y(t,1)}{h^2} &&\approx \frac{Y_{N-1} - 2Y_N + B/A}{h^2} \end{aligned} \tag{6.4}$$

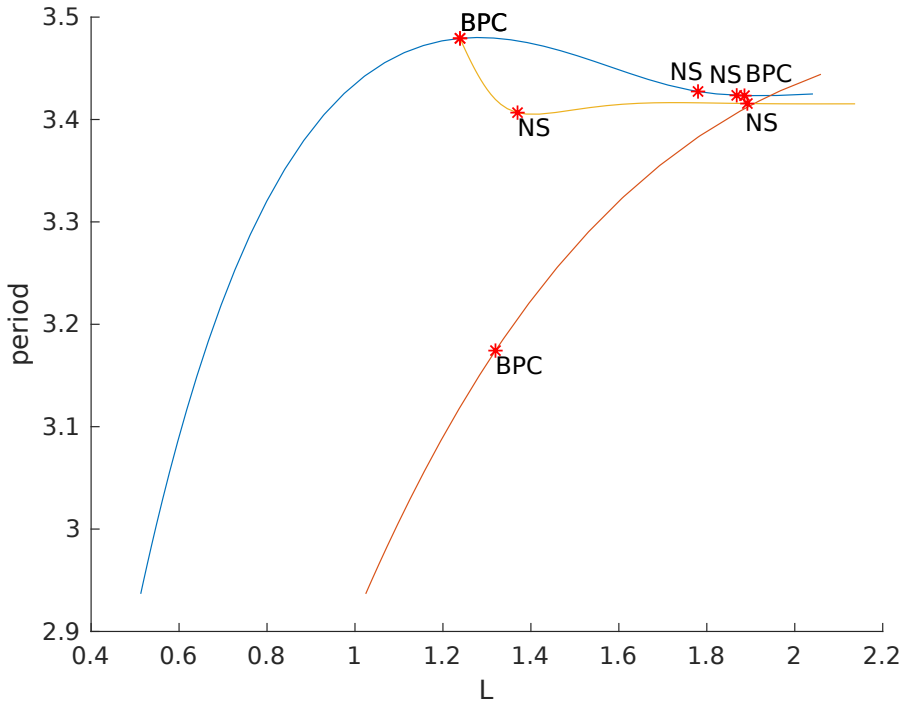
Thus, we have:

$$\begin{aligned}
\frac{\partial X_1}{\partial t} &\approx \frac{D_X}{L^2}\frac{A - 2X_1 + X_2}{h^2} &&+R_{X,1}\\
\frac{\partial Y_1}{\partial t} &\approx \frac{D_Y}{L^2}\frac{B/A - 2Y_1 + Y_2}{h^2} &&+R_{Y,1}\\
\frac{\partial X_N}{\partial t} &\approx \frac{D_X}{L^2}\frac{X_{N-1} - 2X_N + A}{h^2} &&+R_{X,N}\\
\frac{\partial X_N}{\partial t} &\approx \frac{D_Y}{L^2}\frac{Y_{N-1} - 2Y_N + A}{h^2} &&+R_{Y,N}
\end{aligned} \tag{6.5}$$

And for the grid points ih, $i \in \{2, 3, ..., N-1\}$ we have:

$$\begin{aligned}
\frac{\partial X_i}{\partial t} &\approx \frac{D_X}{L^2}\frac{X_{i-1} - 2X_i + X_{i+1}}{h^2} &&+R_{X,i}\\
\frac{\partial Y_i}{\partial t} &\approx \frac{D_Y}{L^2}\frac{Y_{i-1} - 2Y_i + Y_{i+1}}{h^2} &&+R_{Y,i}
\end{aligned} \tag{6.6}$$

We use this discretization of the Brusselator model to study cycles in the Brusselator. To find a cycle we continue the trivial, spatially homogeneous equilibrium $X(t, z) = A$, $Y(t, z) = B/A$ with respect to the parameter L, to find Hopf points at $L \approx 0.5128$, and $L \approx 1.02439$. We continue the cycles that emerge from these Hopf points. This produces the following bifurcation diagram:



The branch B_1 that starts at $L \approx 0.5128$ has a branching point of cycles (BPC), from which another branch of cycles B_{BPC} originates. The branch B_{BPC} was continued as well. It has two Neimark-Sacker bifurcations (NS). The branch B_1 also has two Neimark-Sacker bifurcations and another branching point of cycles. The branch B_2 which starts at $L \approx 1.02439$ has a branching point of cycles and a Neimark-Sacker bifurcation.

The diagram was produced using orthogonal collocation. Cycles in the Brusselator can also be continued using single shooting and multiple shooting with Newton Picard, but branch switching at branching points of cycles in not yet implemented. The diagram above shows similar results as the diagram on page 75 of [19].

6.2 A 1D transport model of a fusion plasma

The second example is about cycles in a finite difference discretization of a one dimensional transport model of a fusion plasma in TOKAMAK reactor. The model was originally proposed in [21]. Continuation of equilibria, and bifurcation of equilibria in this model was already included in the CL_MATCONTL tutorials. The following description of the model is copied verbatim from the CL_MATCONTL tutorials.

The transport model consists of three coupled PDEs describing the particle density n, the plasma temperature T and the radial electric field or poloidal rotation Z:

$$\frac{\partial n}{\partial t} = -\frac{\partial \Gamma}{\partial x} \qquad \Gamma = -D(Z)\frac{\partial n}{\partial x}, \tag{6.7}$$

$$\frac{\partial U}{\partial t} = -\frac{\partial q}{\partial x} \qquad q = -\chi(Z)n\frac{\partial T}{\partial x} + \frac{\Gamma T}{\gamma - 1}, \tag{6.8}$$

$$\varepsilon\frac{\partial Z}{\partial t} = \mu\frac{\partial^2 Z}{\partial x^2} + c_n\frac{T}{n^2}\frac{\partial n}{\partial x} + \frac{c_T}{n}\frac{\partial T}{\partial x} + G(Z), \tag{6.9}$$

where

$$D(Z) = D_0 + D_1\tanh(Z), \tag{6.10}$$

$$\chi(Z) = \frac{1}{(\gamma-1)\zeta}D(Z), \tag{6.11}$$

$$G(Z) = a - b(Z - Z_S) - (Z - Z_S)^3. \tag{6.12}$$

The internal energy U is related to temperature T and density n by

$$U := \frac{nT}{\gamma - 1}. \tag{6.13}$$

The following boundary conditions are used for the problem

$$\Gamma|_{x=\infty} = \Gamma_\infty, \qquad q|_{x=\infty} = q_\infty, \qquad \left.\frac{\partial Z}{\partial x}\right|_{x=\infty} = 0, \tag{6.14}$$

$$\left.\frac{\partial n}{\partial x}\right|_{x=0} = \frac{n}{\lambda_n}, \qquad \left.\frac{\partial T}{\partial x}\right|_{x=0} = \frac{T}{\lambda_T}, \qquad \left.\frac{\partial^2 Z}{\partial x^2}\right|_{x=0} = 0. \tag{6.15}$$

We use a finite difference discretization on the nonuniform grid

$$\Theta := \{0 = x_0 < x_1 < x_2, \ldots, < x_N = L\}, \qquad x_i = L\,(i/N)^3\,. \tag{6.16}$$

The finite difference formulas can be found in (for example) [22]. This leads to a system of $n = 3(N-1)$ ODEs.

The parameter values at the starting point are $\Gamma_\infty = -0.8$, $q_\infty = -0.72$, $D_0 = 1.9$, $D_1 = -1.1$, $a = -1$, $b = -0.3$, $\zeta = 1.1$, $\mu = 0.05$, $\varepsilon = 0.05$, $Z_S = 0$, $\gamma = 5/3$, $\lambda_n = 1.25$, $\lambda_T = 1.5$, $c_n = 1.1$, and $c_T = 0.9$. In this example, we use $N = 50$ so the size of the system is $n = 3(N-1) = 147$.

The cycle, from which we start the continuation, is found by integrating from the initial point of which all coordinates of equal 1. From there, we continue the cycle w.r.t. the parameter q_∞. When starting the continuation with increasing q_∞ from $q_\infty = 0.72$, we find a three limit points of cycles (LPC) and two period doubling points (PD). Note that the second LPC occurs almost immediately after the first PD. After the second period

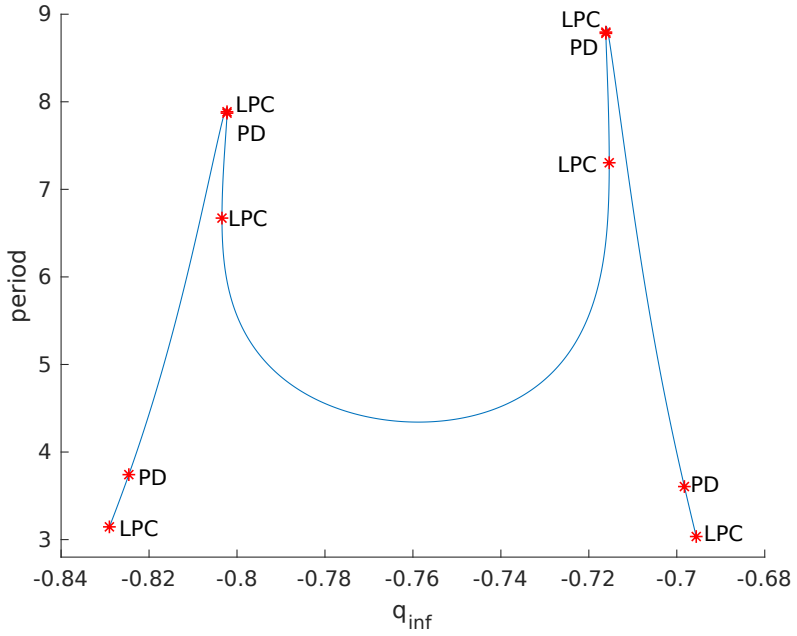


Figure 6.1: Continuation of a cycle in the plasma-system for $N = 50$

doubling point, the continuation goes on for a few more steps and then reverses direction a limit point of cycles. The same sequence of bifurcations is also seen when starting the continuation with decreasing q_∞ from $q_\infty = -0.72$. The continuation of that produced this diagram was computed using orthogonal collocation. We used $n_{tst} = 40$ mesh intervals to continue this branch of cycles. The computation ran on Matlab R2019a and a Intel(R) Core(TM) i7-8750H CPU running at 2.20GHz, and takes about 2 hours. The continuation of this same branch was also performed for $N = 75$ for $q_\infty \in [-0.72, -0.69]$. For $N = 75$ the size of the system of ODEs is 223.

The branch of cycles in figure 6.2 can be continued with multiple shooting as well. However, the size of the system is not so large that the Newton Picard method to leads to time savings. Rather, continuation with Newton-Picard and multiple shooting takes much longer than orthogonal collocation in this system for N up to 75. Continuation of this branch with single shooting will fail, since at some point the cycles become very unstable. In fact, the leading Floquet multiplier will become larger than 10000.

In [23], it was stated that a limit cycle was observed near a saddle-focus homoclinic bifurcation, for $q_\infty = -0.7$, $a = -1$, $b = -0.3$, and $N = 50$. A plot of the Z_0-coordinate of this cycle is shown in figure 6.2. This cycle can be continued using orthogonal collocation with CL_MATCONTL. Due to the complicated shape of the cycle, $n_{tst} = 100$ mesh intervals were used.

The result of the continuation is shown in figure 6.2. The continuation produces a closed curve with 4 limit points of cycles (LPC) and 4 period doubling points (PD). Three of the four period doubling points are very close to a limit point of cycles. It would be interesting to see if this cycle can be continued to a point nearer to the homoclinic to saddle-focus, and how well the continuation methods would cope with an increasingly complicated cycle.

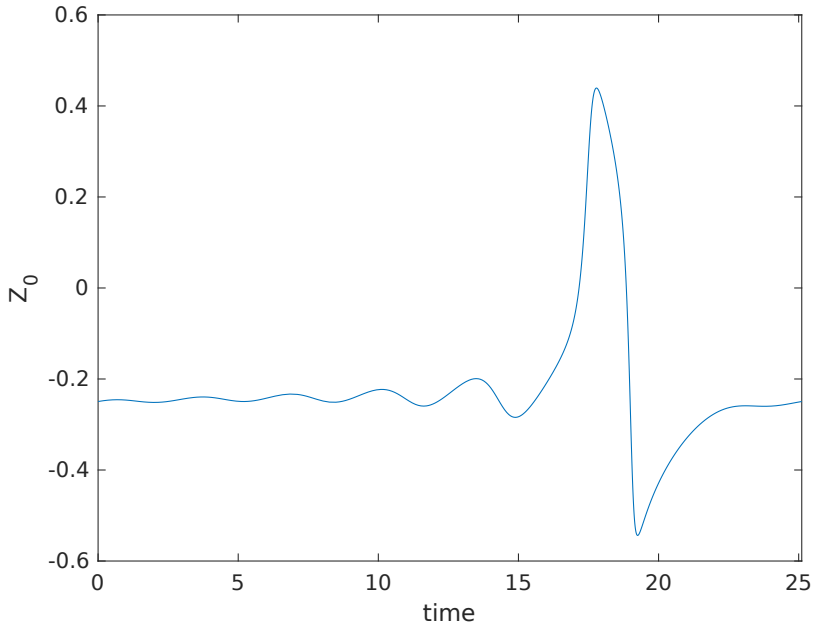


Figure 6.2: Periodic orbit near a saddle-focus homoclinic orbit (at $q_\infty = 0.7, b = -0.3$, and $a = -1$ for $N = 50$ grid points)

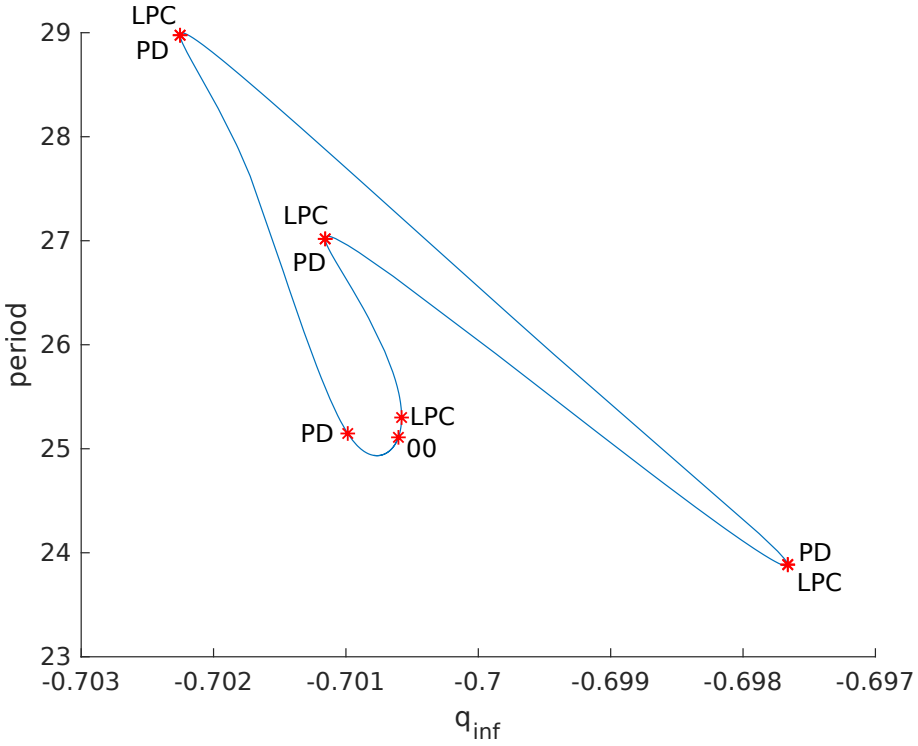


Figure 6.3: Continuation of the cycle shown in figure 6.2

6.3 The nonadiabatic tubular reactor

As a third example, we consider the model:

$$\begin{cases} \dfrac{\partial y}{\partial t} = \dfrac{1}{P_{em}}\dfrac{\partial^2 y}{\partial x^2} - \dfrac{\partial y}{\partial x} - Dy\exp\left[\gamma\left(1-\dfrac{1}{\gamma}\right)\right] \\ \dfrac{\partial \theta}{\partial t} = \dfrac{1}{P_{eh}}\dfrac{\partial^2 \theta}{\partial x^2} - \dfrac{\partial \theta}{\partial x} - \beta(\theta-\theta_0) - BDy\exp\left[\gamma\left(1-\dfrac{1}{\gamma}\right)\right] \end{cases} \tag{6.17}$$

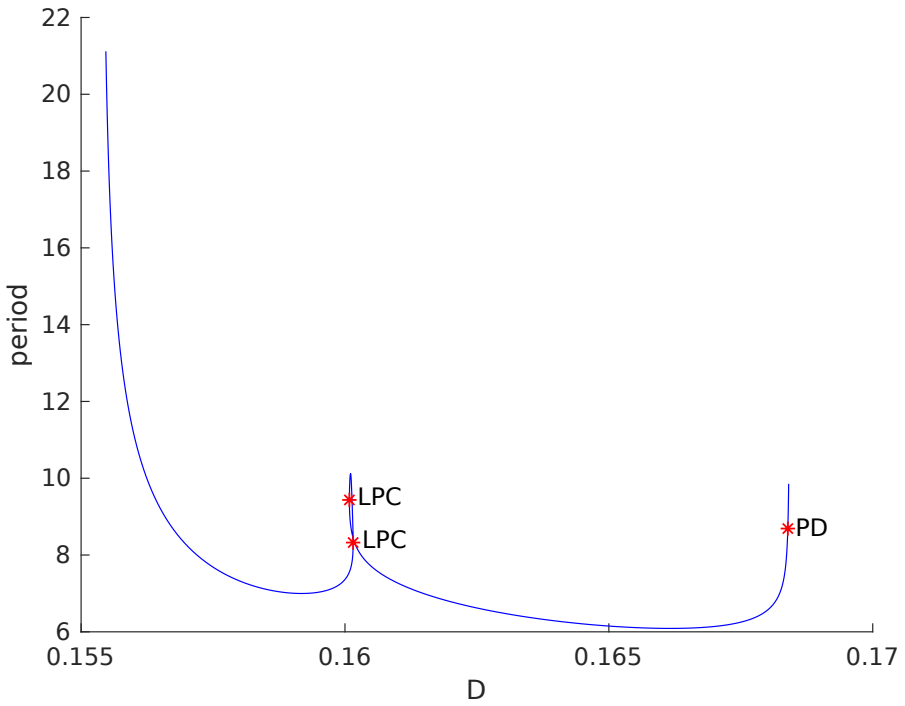
with boundary conditions:

$$y_x(0,t) = P_{em}(y-1), \quad \phi_x(0,t) = P_{eh}(\phi-1), \quad y_x(1,t) = \phi_x(1,t) = 0$$

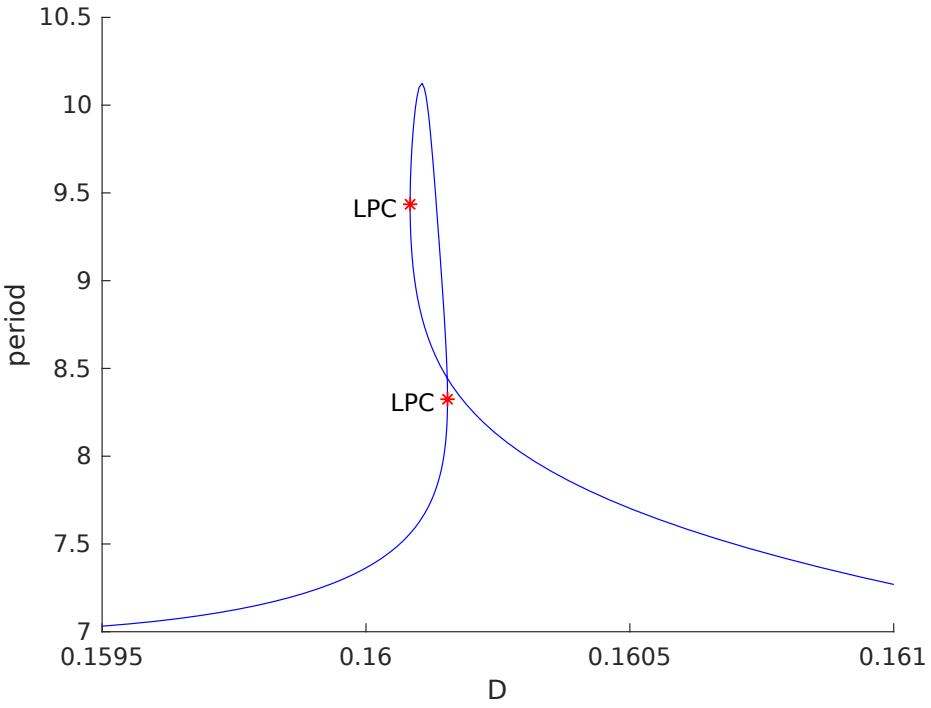
A limit cycle in this model was described in section 10.1.6 of [22]. The cycle was found by time integration from the initial conditions $y(x,0) = 1$, $\phi(x,0) = 1$ for all $x \in [0,1]$ and parameter values:

$$D = 0.165, \quad P_{em} = 5, \quad P_{eh} = 5, \quad \beta = 2.35, \quad \phi_0 = 1, \quad \gamma = 25, \quad B = 0.5$$

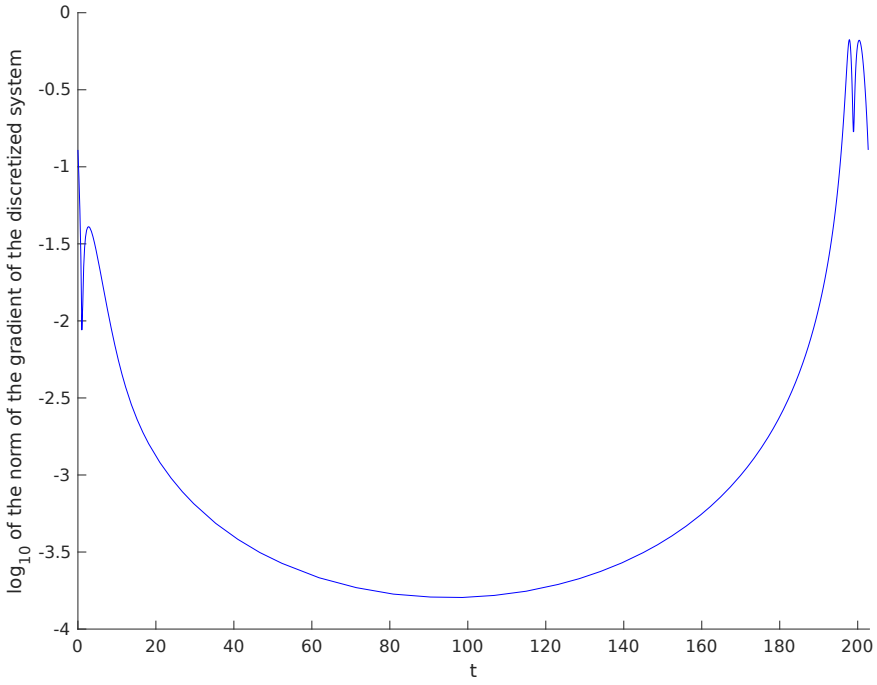
This model was discretized using a equidistant mesh. This limit cycle was continued w.r.t. D to produce the following bifurcation diagram:



Near $D = 0.16$ there are two limit points of cycles:



Near $D = 0.155$ the cycle appears to converge to a homoclinic orbit. This can be seen in the plot below. In the plot we show the base 10 logarithm of the norm of the gradient of the discretized system along the cycle. There is one interval of time for which the norm of the gradient is significantly smaller than any other time period along the cycle. This indicates that the cycle is converging to a homoclinic orbit.



Chapter 7

Conclusion

The Single Shooting-Newton-Picard and Multiple Shooting-Newton-Picard algorithms have been implemented successfully. The algorithms will be made publicly available in the CL_MATCONTL package, along with new tutorials that show how to use them [4]. Up to now, no Matlab program was available, that implemented these algorithms. These methods should be valuable, since they reduce the amount of computations by only considering the most unstable modes of a cycle explicitly, and considering the other, more stable modes, by an iterative method.

In particular, using Newton-Picard with single shooting, one can continue the stable cycle emerging from the Hopf point at $L \approx 0.5$ in the Brusselator with 1000 mesh points, leading to a system of 2000 ODEs. The first step takes less than one minute and uses less than 3GB of memory. In contrast, orthogonal collocation will will run out of memory at 400 mesh points on a PC with 8GB of RAM. As shown by equation 2.10 the memory use of collocation is still quadratic in the size of the system n_{ODE}, whereas for Newton-Picard methods, if the number of nonzero's in Jacobian matrix of the system of ODEs is linear in n_{ODE}, and the amount of memory used by applying the inverse action of this matrix is also linear, then the amount of memory used by Newton-Picard methods is linear in n_{ODE}.

Multiple shooting with Newton-Picard is an essential extension of single shooting with Newton-Picard. This is demonstrated by the fact that the cycles is the plasma model become too unstable for single shooting. Hence, if such extremely unstable cycles are to be analyzed in really large systems, Multiple Shooting with Newton-Picard is essential.

However, orthogonal collocation also performs very well. It is our estimation that is system with up to about 400 to 500 equations, orthogonal collocation will be faster. However, the multipliers obtained with the methods of section 2.1.2 orthogonal collocation are less accurate than the multipliers computed for the shooting methods if the number of mesh intervals is low. In particular, with many mesh points, many real multipliers just below one will appear.

The question in which cases Newton-Picard methods are superior to collocation, is something which requires more research. In particular, a large system with cycles with few unstable modes, is likely to be a good candidate to demonstrate the practical value of Newton-Picard methods.

If Newton-Picard methods are to be applied, it would be very helpful to use a Matlab cluster to speed up to computations, since for systems of hundreds of equations the computations can take hours or even days on a single computer. Some parts of the single and multiple shooting code can already be run in parallel, but more parallelization should be added. Especially in multiple shooting there are many simple loops over all the mesh intervals that can be readily parallelized, in the sense that there are no conceptual changes

to the algorithm are needed. The parallelization does require some programming effort, since global variables are (understandably) not distributed over the entire cluster of Matlab workers, which requires that all variables in a loop, that is to be parallelized, to be copied to local variables.

Continuation of bifurcations of limit cycles can also be considered. In chapter 7 his PhD thesis [19], K. Lust outlines methods for continuing period doubling bifurcations, Neimark-Sacker bifurcations, and limit point of cycles in large systems of using Newton-Picard methods. For period doubling bifurcations Lust has implemented and tested his method [24, 25]. Hence, it seems that continuation of bifurcations of limit cycles in large systems of ODEs is a relatively simple task of implementing Lust' methods.

Bibliography

[1] V. A. Selivanov, M. Cascante, M. Friedman, M. F. Schumaker, M. Trucco, and T. V. Votyakova, "Multistationary and oscillatory modes of free radicals generation by the mitochondrial respiratory chain revealed by a bifurcation analysis," *PLoS computational biology*, vol. 8, no. 9, p. e1002700, 2012.

[2] D. Bindel, M. Friedman, W. Govaerts, J. Hughes, and Y. A. Kuznetsov, "Numerical computation of bifurcations in large equilibrium systems in matlab," *Journal of Computational and Applied Mathematics*, vol. 261, pp. 232–248, 2014.

[3] L. Dieci and M. J. Friedman, "Continuation of invariant subspaces," *Numerical Linear Algebra with Applications*, vol. 8, no. 5, pp. 317–327, 2001.

[4] M. Pekkér, "CL_MATCONTL." `https://www.uah.edu/science/departments/math/373-faculty/pekkermj/9996-software-development`, 2019. accessed May 9, 2019.

[5] H. Meijer, F. Dercole, and B. Oldeman, "Numerical bifurcation analysis." article in Encyclopedia of Complexity and Systems Science, part 14, pages 6392-6352, 2009.

[6] C. De Boor and B. Swartz, "Collocation at Gaussian points," *SIAM Journal on Numerical Analysis*, vol. 10, no. 4, pp. 582–606, 1973.

[7] E. J. Doedel, W. Govaerts, Y. A. Kuznetsov, and A. Dhooge, "Numerical continuation of branch points of equilibria and periodic orbits," *International Journal of Bifurcation and Chaos*, vol. 15, no. 03, pp. 841–860, 2005.

[8] W. Mestrom, "Continuation of limit cycles in MATLAB," Master's thesis, Utrecht University, the Netherlands, 2002.

[9] A. Dhooge, W. Govaerts, Y. A. Kuznetsov, W. Mestrom, and A. Riet, "Cl_matcont: a continuation toolbox in matlab," in *Proceedings of the 2003 ACM symposium on Applied computing*, pp. 161–166, ACM, 2003.

[10] E. Doedel, "Auto Website." `http://indy.cs.concordia.ca/auto/`, 2019. accessed May 16, 2019.

[11] A. Dhooge, W. Govaerts, Y. A. Kuznetsov, H. G. E. Meijer, and B. Sautois, "New features of the software matcont for bifurcation analysis of dynamical systems," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 14, no. 2, pp. 147–175, 2008.

[12] K. Lust, "Improved numerical floquet multipliers," *International Journal of Bifurcation and Chaos*, vol. 11, no. 09, pp. 2389–2410, 2001.

[13] A. W. Bojanczyk, G. H. Golub, and P. Van Dooren, "Periodic schur decomposition: algorithms and applications," in *Advanced Signal Processing Algorithms, Architectures, and Implementations III*, vol. 1770, pp. 31–43, International Society for Optics and Photonics, 1992.

[14] D. Kressner, "An efficient and reliable implementation of the periodic qz algorithm," *IFAC Proceedings Volumes*, vol. 34, no. 12, pp. 183–188, 2001.

[15] T. F. Fairgrieve and A. D. Jepson, "Ok floquet multipliers," *SIAM journal on numerical analysis*, vol. 28, no. 5, pp. 1446–1462, 1991.

[16] D. Roose, K. Lust, A. Champneys, and A. Spence, "A Newton-Picard shooting method for computing periodic solutions of large-scale dynamical systems," *Chaos, Solitons & Fractals*, vol. 5, no. 10, pp. 1913–1925, 1995.

[17] K. Lust and D. Roose, "An adaptive Newton–Picard algorithm with subspace iteration for computing periodic solutions," *SIAM Journal on Scientific Computing*, vol. 19, no. 4, pp. 1188–1209, 1998.

[18] K. Lust and D. Roose, "Computation and bifurcation analysis of periodic solutions of large-scale systems," in *Numerical methods for bifurcation problems and large-scale dynamical systems*, pp. 265–301, Springer, 2000.

[19] K. Lust, *Numerical bifurcation analysis of periodic solutions of partial differential equations*. PhD thesis, K.U.Leuven, 1997.

[20] J. Sánchez and M. Net, "On the multiple shooting continuation of periodic orbits by newton–krylov methods," *International Journal of Bifurcation and Chaos*, vol. 20, no. 01, pp. 43–61, 2010.

[21] H. Zohm, "Dynamic behavior of the L–H transition," *Physical review letters*, vol. 72, no. 2, p. 222, 1994.

[22] W. Govaerts, *Numerical methods for bifurcations of dynamical equilibria*. Philadelphia: SIAM, 2000.

[23] H. De Blank, Y. A. Kuznetsov, M. Pekkér, and D. Veldman, "Degenerate Bogdanov-Takens bifurcations in a one-dimensional transport model of a fusion plasma," *Physica D: Nonlinear Phenomena*, vol. 331, pp. 13–26, 2016.

[24] K. Engelborghs, K. Lust, and D. Roose, *A Newton-Picard method for accurate computation of period doubling bifurcation points of large-scale systems of ODEs*. Katholieke Universiteit Leuven. Departement Computerwetenschappen, 1996.

[25] K. Engelborghs, K. Lust, and D. Roose, "Direct computation of period doubling bifurcation points of large-scale systems of ODEs using a Newton-Picard method," *IMA Journal of Numerical Analysis*, vol. 19, no. 4, pp. 525–547, 1999.