Collocation
a method for computing periodic solutions of ordinary
differential equations

N.J. Mallon
N.J.Mallon@student.tue.nl
Department of mechanical engineering
Eindhoven University of Technology

DCT report 2002.35
May 2002
Preface

This report is the result of an internal traineeship at the Dynamics and Control group from the Department of Mechanical Engineering of the Technical University of Eindhoven. The timescale for the traineeship was 6 weeks but due the birth of my beautiful baby daughter Evi on the 10th of march this time was spread over a period from December 2001 until the end of May 2002. I would like to thank dr. ir. Remco I. Leine for his support and supervision during this project.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>ii</td>
</tr>
<tr>
<td>Summary</td>
<td>vi</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Collocation Theory</td>
<td>3</td>
</tr>
<tr>
<td>2.1 The Basic Idea of Collocation</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Collocation Theory for Solving Non-Linear Boundary Value Problems</td>
<td>4</td>
</tr>
<tr>
<td>2.2.1 Quasi-Linearization</td>
<td>7</td>
</tr>
<tr>
<td>2.2.2 Elimination of Local Unknowns</td>
<td>8</td>
</tr>
<tr>
<td>2.2.3 The Collocation Equations</td>
<td>9</td>
</tr>
<tr>
<td>2.2.4 Order and Convergence of the Collocation Method</td>
<td>10</td>
</tr>
<tr>
<td>3 Computing Periodic Solutions by Collocation</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Theory for Solving the Two Point BVP</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Stability of the Periodic Solution</td>
<td>13</td>
</tr>
<tr>
<td>4 Numerical Results</td>
<td>15</td>
</tr>
<tr>
<td>5 Conclusions</td>
<td>21</td>
</tr>
<tr>
<td>Further Research</td>
<td>22</td>
</tr>
<tr>
<td>Bibliography</td>
<td>24</td>
</tr>
<tr>
<td>A Matlab Implementation</td>
<td>27</td>
</tr>
</tbody>
</table>
Summary

In this report the method of collocation for computing periodic solutions is being studied. Collocation stands for an discretization of the solution of a set of ordinary differential equations at certain meshpoints which satisfies the set of ODEs at certain preselected collocations points. We showed that with this condition and the application of a quadrature scheme the approximation could be defined with the same order of accuracy as the quadrature scheme applied. For solving the resulting set of non-linear equations we first used a technique called quasi-linearization to solve the non-linear boundary value problem in a sequence of linear BVPs. For elimination of the local unknowns we presented a parameter condensation technique which could be applied to the linear problem. These steps resulted in a scheme which could be applied for any given quadrature scheme and therefore for any given order of accuracy.

For computing periodic solutions we showed that this problem could be defined as a two point BVP problem with an extra unknown parameter. By rewriting this problem in a dimensionless form we could use the same steps as followed for the regular BVP problem which resulted in an analogous scheme.

With the implementation of the collocation scheme for computing periodic solutions in MATLAB the performance of the method could be compared with two other methods, the finite difference method and the single shooting method. For the implementation we applied the Gaussian quadrature scheme which is the quadrature scheme of the highest order. By two numerical testcases we illustrated that the method of collocation is of excellent accuracy and is very robust for bad initial guesses.
Chapter 1

Introduction

Computation of periodic solutions of a set of non-linear ordinary differential equations is used for studying the behavior of non-linear dynamical systems. Various numerical algorithms, also known as periodic solution solvers, exist for finding periodic solutions. Well known solvers are the single- or multiple shooting method and the finite difference method [4].

AUTO is a software package for continuation and bifurcation problems in ordinary differential equations. Periodic solutions are computed in this package by means of a method called collocation [1]. In this report we want to find out in detail what the basic principle is of the method of collocation and how the method can be implemented for computing periodic solutions of a set of non-linear ordinary differential equations. For a performance insight we also would like to know what the main advantages and/or drawbacks are in comparison with the known solvers.

In the first chapter we explain the basic idea of collocation and show how this idea can be applied for solving non-linear boundary value problems. In the next chapter it will be shown that the problem of computation of periodic solutions can be formulated as a non-linear boundary value problem and it will be shown how the method of collocation may be applied to solve this problem. In chapter 4 we setup a numerical test to compare the method of collocation with the single shooting method and the finite difference method. Finally the results are discussed in the chapter 5.
Chapter 2

Collocation Theory

In this chapter we will introduce the method of collocation and show how the method of collocation can be applied for solving non-linear boundary value problems. Consider the set of ordinary non-linear autonomous differential equations

\[ x' = f(x) \]  

(2.1)

where \( x \) is the \( n \) dimensional state vector, \( f(x) \) a set of \( n \) first order differential equations and the prime (\( ' \)) denotes differentiation with respect to time \( t \). Due to the non-linear nature of (2.1) analytical techniques for finding a closed form solution \( x(t) \) for the set of equations are not feasible. One must therefore consider approximations \( x_\pi(t) \) for the solution of (2.1). These approximations can be computed by means of numerical methods which results in a discretisation for the solution. Discretisation means that the solution is represented by function values on so called mesh points which are mutual connected by interpolation functions. There are various methods for defining these interpolation functions. In the next section we will present the basic idea of one of these methods which is called the method of collocation.

2.1 The Basic Idea of Collocation

The method of collocation requires that the approximate solution satisfied the ordinary differential equations at certain preselected collocation points. For illustration of the method, we start with a simple example for which we consider the following non-autonomous ordinary differential equation

\[ x' = 3t^2, \text{ with initial condition } x(0) = 1. \]  

(2.2)

The exact solution of (2.2) is \( x(t) = 1 + t^3 \). We want to approximate this solution on the range \( 0 \leq t \leq 1 \) for which we will use a first-degree polynomial, described by

\[ x_\pi(t) = a_0 + a_1 t. \]  

(2.3)

Requiring that the initial condition to be satisfied at \( t = 0 \) gives the trivial solution \( a_0 = 1 \). Substitution from (2.3) into (2.2) results in an expression in the coefficient \( a_1 \) and the independent variable \( t \).
CHAPTER 2. COLLOCATION THEORY

\[ a_1 = 3t^2. \quad (2.4) \]

Requiring that the ODE is satisfied at the interior collocation point \( t = 0.5 \) gives the numerical value for the last unknown coefficient, \( a_1 = 0.75 \). The corresponding approximating polynomial becomes

\[ x_π(t) = 1 + 0.75t. \quad (2.5) \]

The approximate and exact solution are plotted in Figure 2.1. The Figure shows the basic idea of collocation, computing a solution which satisfies the initial conditions, and the differential equation at preselected collocation points. This basic idea can easily be applied to any initial or boundary value problem with an arbitrary number of differential equations and order of (piecewise) approximation polynomial. In the next section the basic idea of collocation will be applied to non-linear boundary value (BVP) problems. It will be shown that the placement of the collocation points is not arbitrary and that the solution of the problem can be computed by a Newton type of method known as quasi-linearization.

2.2 Collocation Theory for Solving Non-Linear Boundary Value Problems

In this section we will show how the method of collocation can be used to solve non-linear autonomous boundary value problems of the form (2.6)

\[ x' = f(x) \text{ with } g(x(a), x(b)) = 0. \quad (2.6) \]
2.2. COLLOCATION THEORY FOR SOLVING NON-LINEAR BOUNDARY VALUE PROBLEMS

An approximate solution of the problem (2.6) by collocation on a subinterval in \( a \leq t \leq b \), denoted by \([t_i, t_{i+1}]\), satisfied the following interpolation conditions:

\[
x_{\pi}(t_i) = x_i \text{ and } x_{\pi}'(t_{ij}) = f(x_{ij}) \text{ for } 1 \leq i \leq N, 1 \leq j \leq k,
\]

with \( t_{ij} \) the \( j^{th} \) collocation point in subinterval \( i \) and \( x_{\pi}(t) \) a polynomial of order \( k \). Figure 2.2 shows the locations of the intervals and collocation points.

The approximation \( x_{\pi}(t) \) can be written in terms of its initial condition and its first derivative as shown in (2.8)

\[
x_{\pi}(t) = x_i + \int_{t_i}^{t} x_{\pi}'(\xi) d\xi.
\]

If the approximation satisfies the interpolation conditions (2.7) then at the \( k \) collocation points the derivative \( f(x) \) is equal to the derivative \( x_{\pi}' \). With this information we can define an interpolation polynomial of degree \( k - 1 \) for the state equations. A standard way to define the interpolation polynomial is the Lagrange form:

\[
p(t) = \sum_{l=1}^{k} f(x_{il}) L_l(\rho) \quad \text{with } \rho = \frac{t - t_i}{h_i}, x_{il} = x_{\pi}(t_{il}) \text{ and } \rho \in [0, 1]
\]

and

\[
L_l(\rho) = \prod_{j=1}^{k} \frac{\rho - \rho_j}{\rho_l - \rho_j}, \quad \rho_l = \frac{t_{il} - t_i}{h_i}, \quad h_i = t_{i+1} - t_i.
\]

For the polynomials \( L_l(\rho) \) we have rescaled time \( t \) for each interval \( i \) to the domain \([0, 1]\) so that they are for each interval similar. Since the polynomials satisfy

\[
L_i = \delta_{ij} \text{ with } \delta_{ij} = \begin{cases} 1 & \text{for } j = i \\ 0 & \text{for } j \neq i \end{cases}
\]
it is easy to see that the polynomial \( p(t) \) satisfies \( f(x_{ij}) = p(t_{ij}) \). With \( p(t) \) the state equations are now no longer a function of the state variables \( x \) so the integration in (2.8) can be done straight forward. Substitution of (2.9) in to (2.8) yields

\[
x_{\pi}(t) = x_i + \int_{t_i}^t \sum_{l=1}^k f(x_{il})L_l(\rho)\,d\rho.
\]  

(2.12)

Now we replace the integral with a numerical quadrature method and use the (scaled\(^1\)) integration points as collocation points. Now we arrive at two equations, the first one (2.13) for the local unknowns, the values \( x_{il} \) for determining the derivatives \( f(x_{il}) \), and the second one (2.14) for the global unknowns, the values \( x_i \).

\[
x_{\pi}(t_{ij}) = x_i + h_i \sum_{l=1}^k \alpha_{jl} f(x_{il}) = x_{ij} \quad \text{with} \quad \alpha_{jl} = \int_0^1 L_l(\rho)\,d\rho
\]  

(2.13)

\[
x_{\pi}(t_{i+1}) = x_i + h_i \sum_{l=1}^k \beta_l f(x_{il}) = x_{i+1} \quad \text{with} \quad \beta_l = \int_0^1 L_l(\rho)\,d\rho
\]  

(2.14)

If we now extend the polynomial to the subinterval \([t_{i+1}, t_{i+2}]\) by using (2.14) with \( i \) replaced by \( i + 1 \), the piecewise polynomial will be continuously matched at \( t_{i+1} \). Extending to all \( 1 \leq i \leq N \), we obtain a continuous piecewise polynomial function of order \( k \) over \([a, b]\) which satisfies the ODE at the collocations points \( t_{ij} \).

The final equation (2.14) has the identical form as the Runge-Kutta scheme. Due to the fact that the function is based on a numerical quadrature method the final scheme has an implicit character. From this we can conclude that the collocation scheme is identical to a one-step implicit Runge-Kutta scheme. For illustration of this implicitness we apply Simpson’s quadrature scheme which approximates integration over the interval \([0, 1]\), and has the following set of parameters \( k = 3 \), \( \rho_1 = 0 \), \( \rho_2 = 0.5 \) and \( \rho_3 = 1 \). The Lagrange interpolation functions for this parameters are defined by

\[
L_1(\rho) = \frac{(\rho - 0.5)(\rho - 1)}{0.5}, \quad L_2(\rho) = \frac{(\rho - 0)(\rho - 1)}{-0.25}, \quad L_3(\rho) = \frac{(\rho - 0)(\rho - 0.5)}{0.5}.
\]

The three polynomials are plotted in Figure 2.3. From Figure 2.3 it can noted that the polynomials satisfy property (2.11). With these polynomials we compute the values \( \alpha_{jl} \) and \( \beta_l \) as shown in (2.13) and (2.14). The numerical values for \( \alpha \) and \( \beta \), together with the (scaled) locations of the collocation points \( \rho_l \) are often presented in a so called Butcher-array, which is for this scheme shown in Table 2.1.

From Table 2.1 we can see the problem which arises if we want to solve equation (2.13). For calculation of for example \( f(x_{i2}) \) we first need \( f(x_{i3}) \) and vice versa. Solving this non-linear equation would require an expensive Newton method for each

\(^1\)Integration points of quadrature methods which approximate an other domain as \([0, 1]\) such as Gauss integration are scaled to the \([0,1]\) domain.
2.2. COLLOCATION THEORY FOR SOLVING NON-LINEAR BOUNDARY VALUE PROBLEMS

Figure 2.3: Lagrange polynomials.

Table 2.1: Butcher array for Simpson’s quadrature scheme.

<table>
<thead>
<tr>
<th>$\rho_l$</th>
<th>$\alpha_{11}$</th>
<th>$\ldots$</th>
<th>$\alpha_{13}$</th>
<th>$\alpha_{13}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>5/24</td>
<td>1/3</td>
<td>-1/24</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>1</td>
<td>1/6</td>
<td>2/3</td>
<td>1/6</td>
<td>$\alpha_{33}$</td>
</tr>
<tr>
<td>$\beta_l$</td>
<td>1/6</td>
<td>2/3</td>
<td>1/6</td>
<td></td>
</tr>
</tbody>
</table>

subinterval. This brings us at point how the equations (2.13) and (2.14) or perhaps only the equation of interest (2.14) are going to be solved. For this purpose we will introduce two methods, first the method of quasi-linearization for rewriting the solution of the non-linear BVP into a sequence of solutions of a linear BVP, and second a parameter condensation procedure which can be applied to linear problems to eliminated the local unknowns.

2.2.1 Quasi-Linearization

Newton’s method for algebraic equations is obtained by expanding the functions in Taylor series and truncating the non-linear terms at each iteration. The quasi-linearization method does the same for the non-linear boundary value problem (2.6). For this we rewrite the problem for small perturbations $w$ from an initial guess $x^m(t)$

\begin{align}
(x^m(t) + w)' &= f(x^m(t)) + \frac{\partial f}{\partial x}(x^m(t))w \\
\quad w' &= A(t)w + q(t) = f^*(t, w)
\end{align}

(2.15) (2.16)

where

\begin{align}
A(t) &= \frac{\partial f}{\partial x}(x^m(t)) \quad \text{and} \quad q(t) = f(x^m(t)) - (x^m(t))'.
\end{align}

(2.17)
And the boundary conditions as

\[ g(x(a), x(b)) = 0 \rightarrow B_aw_1 + B_bw_N = -g(x^m(a), x^m(b)) \]  

\[ (2.18) \]

with

\[ B_a = \frac{\partial g(x(a), x(b))}{\partial x}(x^m(a)) \quad \text{and} \quad B_b = \frac{\partial g(x(a), x(b))}{\partial x}(x^m(b)). \]  

\[ (2.19) \]

The method has the identical form as the Newton method, a residual \( q(t) \) in the form of the difference between the ODE and the derivative of the solution and a jacobian operator acting on the correction step \( w \). The residual in this case is zero if the solution satisfies the ODE at the collocation points. The solution of the linearized BVP is the solution of the collocation equations for the new set of variables \( w_i \) which are analogous to (2.13) and (2.14).

\[ w(t_{ij}) = w_i + h_i \sum_{l=1}^{k} \alpha_{jl} f^*(t_{il}, w_{il}) = w_{ij} \]  

\[ (2.20) \]

\[ w(t_{i+1}) = w_i + h_i \sum_{l=1}^{k} \beta_{il} f^*(t_{il}, w_{il}) = w_{i+1} \]  

\[ (2.21) \]

The coefficients in the linear problem all depend on the previous iteration \( x^m(t) \). The quasi-linearization procedure therefore defines a sequence of linear BVPs which solutions will converge to the solution of the non-linear problem, given that the initial guess was close enough to it [2]. Thus, if we now find a way to solve these equations, then we obtain a method for solving non-linear BVP as well.

### 2.2.2 Elimination of Local Unknowns

For the linear BVP (2.16) we now will present a procedure to eliminate the local unknowns in equation (2.21), the derivatives \( f^*(t_{il}, w_{il}) \) where the \((*)\) denotes the derivatives for the set of variables \( w_i \). This method is also known as *parameter condensation* in finite element literature. Substitution of (2.21) in (2.16) yields

\[ f^*_{ij} = f^*(t_{ij}, w_i + h_i \sum_{l=1}^{k} \alpha_{jl} f^*(t_{il}, w_{il})) \]

\[ = A(t_{ij})w_i + A(t_{ij})h_i \sum_{l=1}^{k} \alpha_{jl} f^*(t_{il}, w_{il}) + q(t_{ij}) \]  

\[ (2.22) \]

And in vector form for \( 1 \leq j \leq k \) as

\[
\begin{bmatrix}
  f^*_{11} \\
  \vdots \\
  f^*_{ik}
\end{bmatrix}
= \begin{bmatrix}
  A(t_{i1}) \\
  \vdots \\
  A(t_{ik})
\end{bmatrix}
\begin{bmatrix}
  w_i + h_i \\
  \vdots \\
  w_i + h_i
\end{bmatrix}
+ \begin{bmatrix}
  \alpha_{11} A(t_{i1}) & \cdots & \alpha_{1k} A(t_{i1}) \\
  \vdots & \ddots & \vdots \\
  \alpha_{11} A(t_{ik}) & \cdots & \alpha_{kk} A(t_{ik})
\end{bmatrix}
\begin{bmatrix}
  f^*_{11} \\
  \vdots \\
  f^*_{ik}
\end{bmatrix}
+ \begin{bmatrix}
  q(t_{i1}) \\
  \vdots \\
  q(t_{ik})
\end{bmatrix}
\]

\[ (2.23) \]
The equations of (2.26) together with the boundary conditions (2.18) form a linear set of equations, which is only dependent on the global unknowns $w_i$ and function values of the previous iteration $x^m(t)$.

$$f_i^* = W_i^{-1}V_i w_i + W_i^{-1}q_i,$$

(2.24)

where $W_i \in \mathbb{R}^{nk \times nk}$, $V_i \in \mathbb{R}^{nk \times n}$, $f_i^* \in \mathbb{R}^n$ and $q_i \in \mathbb{R}^n$ are defined by

$$W_i = I - h_i \begin{bmatrix} \alpha_{11}A(t_{i1}) & \cdots & \alpha_{1k}A(t_{i1}) \\ \vdots & \ddots & \vdots \\ \alpha_{1k}A(t_{ik}) & \cdots & \alpha_{kk}A(t_{ik}) \end{bmatrix}$$

$$V_i = \begin{bmatrix} A(t_{i1}) \\ \vdots \\ A(t_{ik}) \end{bmatrix}$$

and $q_i = \begin{bmatrix} q(t_{i1}) \\ \vdots \\ q(t_{ik}) \end{bmatrix}$.

(2.25)

It is obvious that for $h_i$ small enough $W$ is invertible.

### 2.2.3 The Collocation Equations

In the previous sections we formulated the solution for non-linear BVP and presented a strategy to solve the resulting set of equations. The final collocation equations (2.26) follow from the substitution of (2.24) in to (2.21)

$$w_{i+1} = w_i + h_i \sum_{l=1}^{k} \alpha_{jl} f_j^*(t_{il}, w_{il})$$

$$= w_i + h_i \begin{bmatrix} \beta_1I & \cdots & \beta_kI \end{bmatrix} [W_i^{-1}V_i w_i + W_i^{-1}q_i]$$

$$= \Gamma_i w_i + r_i$$

(2.26)

where

$$D = \begin{bmatrix} \beta_1I & \cdots & \beta_kI \end{bmatrix}, \quad \Gamma_i = I + h_i DW_i^{-1}V_i \quad \text{and} \quad r_i = h_i DW_i^{-1}q_i.$$  

(2.27)

The equations of (2.26) together with the boundary conditions (2.18) form a linear set of $n(N+1)$ equations, which need to be solved in each iteration until converge or an other stop criterium has occurred

$$\begin{bmatrix} -\Gamma_1 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -\Gamma_N & I \\ B_a & 0 & 0 & B_b \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_{N+1} \end{bmatrix} = \begin{bmatrix} r_1 \\ \vdots \\ -g(x^m(a), x^m(b)) \end{bmatrix}.$$  

(2.28)

The procedure needs for startup an initial solution profile $x_\pi(t)$ from which the values $x_1, \ldots, x_{N+1}$ and $f_1, \ldots, f_N$ are given, where

$$f_i = \begin{bmatrix} f(x_{i1}) \\ \vdots \\ f(x_{ik}) \end{bmatrix}.$$  

(2.29)
With these values for \( x_i \) and \( f_i \) the values for \( x_{il} \) can be determined with (2.13). Subsequently the values for \( A(x_{il}) \) and \( f(x_{il}) \) can be determined and the set of equations (2.28) can be constructed. After solving the linear system the approximate solution \( x^n(t) \) and the derivatives at the collocation points \( f_i \) must be updated to

\[
\begin{align*}
  x_{\pi}^{m+1}(t_i) &= x_{\pi}^m(t_i) + w_i \\
  f_i^{m+1} &= f_i^m + W^{-1}_i V w_i + W^{-1}_i q_i
\end{align*}
\] (2.30)

(2.31)

from which new values for \( x_{il} \) can be determined.

### 2.2.4 Order and Convergence of the Collocation Method

The solution for \( x_\pi(t) \) is for the collocation method obtained by the Newton’s type of method quasi-linearization. The method converges quadratically provided the initial guess is sufficiently close to \( x(t) \). The order of the collocation method is determined by the order of the applied quadrature method \([3]\). We can therefore conclude that the maximum order for a \( k \)-stage collocation method is \( 2k \). This means that

\[
|x_n - x_{\pi,n}| \leq O(h^{2k}), \quad 1 \leq n \leq N + 1.
\] (2.32)

This order can be attained by using a Gauss quadrature method, or in other words by collocation at Gaussian points. In this case the method is often called Gauss-Legendre collocation, which comes from the fact that the \( k \) Gauss points are the zeros of the Legendre polynomial of order \( k \).
Chapter 3

Computing Periodic Solutions by Collocation

Computing periodic solutions of a set of (autonomous non-linear) ODEs requires to solve the two point boundary value problem

\[ x' = f(x) \text{ with } x(0) = x(T). \]  

(3.1)

This BVP is similar to the problem as presented in (2.6) with the difference that the period time \( T \) brings an extra unknown parameter into the problem. In the next section we will present how this problem can be transformed to an identical form as (2.6) so that the procedure for solving non-linear BVP with the method of collocation can be applied.

3.1 Theory for Solving the Two Point BVP

We want to transform problem (3.1) to the same form as (2.6). For this transformation we first write the problem in a dimensionless form

\[ \frac{dx}{d\tau} = T f(x) \text{ with } x(0) = x(1), \tau = \frac{t}{T} \text{ and } \tau \in [0,1]. \]  

(3.2)

Next the set of \( n \) ODEs is rewritten for both small perturbations \( w \) on the variables \( x_i \) as for a small perturbation \( dT \) on the period time \( T \) from an initial guess \( [x^m(\tau), T^m] \) as described for the method of quasi-linearization.

\[ \frac{dw}{d\tau} = A(\tau)w + f(x^m(\tau))dT + q(\tau) \]  

(3.3)

with

\[ A(\tau) = T^m \frac{\partial f}{\partial x}(x^m(\tau)) \text{ and } q(\tau) = T^m f(x^m(\tau)) - (x^m(\tau))' \]  

(3.4)

and the boundary conditions as

\[ x(0) - x(1) = 0 \rightarrow w_1 - w_{N+1} = x_{N+1}^m - x_1^m. \]  

(3.5)
With (3.3) expression (2.24) for elimination of the local unknowns becomes

\[ f_i = W^{-1}V w_i + W^{-1}U dT + W^{-1}q_i. \]  

(3.6)

with \( f_i, V_i, W_i \) and \( q_i \) analogous to (2.25) and \( U_i \in \mathbb{R}^{nk} \) defined by

\[ U_i = \begin{bmatrix} f(x^m(\tau_{i1})) \\ \vdots \\ f(x^m(\tau_{ik})) \end{bmatrix}. \]  

(3.7)

Substitution of (3.6) into (2.21) results in a new set of \( n \times N \) collocation equations for computation of the periodic solution.

\[ w_{i+1} = w_i + h_i \sum_{l=1}^{k} \alpha_{jl} f(t_l, w_l) \]

\[ = w_i + h_i \begin{bmatrix} \beta_1 I & \cdots & \beta_k I \end{bmatrix} [W_i^{-1}V w_i + W_i^{-1}U_i dT + W_i^{-1}q_i] \]

\[ = \Gamma_i w_i + \Lambda_i dT + r_i \]  

(3.8)

with \( D, \Gamma_i \) and \( r_i \) analogous to (2.27) and

\[ \Lambda_i = h_i DW_i^{-1}U_i. \]  

(3.9)

Since (3.8) together with (3.5) is a system of \( n(N+1) \) equations in \( n(N+1)+1 \) unknowns, it cannot be solved uniquely. The reason for this is that the phase of the periodic solution is not yet fixed. To overcome this problem a phase condition, also known as an anchor equation, is added to the system of equations. A phase condition can be expressed as a fixed point at the periodic solution but this has a couple of drawbacks. A better anchor equation is known as the orthogonality condition

\[ f(x_1)w_1 = 0. \]  

(3.10)

This condition is probably the most popular anchor equation and is robust for inner loops of a periodic solution and yields a relative good convergence without being numerically too expensive [4]. The linear set of equations which must be solved sequentially analogous to (2.28) which follows from (3.5), (3.8) and (3.10) now becomes

\[ \begin{bmatrix} -\Gamma_1 & I & 0 & 0 & -\Lambda_1 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & -\Gamma_N & I & -\Lambda_N \\ I & 0 & 0 & -I & 0 \\ f(x^m_1) & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ \vdots \\ w_{N+1} \\ dT \end{bmatrix} = \begin{bmatrix} r_1 \\ \vdots \\ \vdots \\ r_N \\ x^m_{N+1} - x^m_1 \end{bmatrix}. \]  

(3.11)

After each iteration the approximate solution \( x^m(t) \) must be updated as in (2.30) and \( f_i \) and the period time \( T \) as

\[ f_{i}^{m+1} = f_{i}^m + W_i^{-1}V w_i + W_i^{-1}U_i dT + W_i^{-1}q_i \]  

(3.12)

\[ T_{m+1} = T_{m} + dT \]  

(3.13)
3.2 Stability of the Periodic Solution

To access the stability properties of a periodic solution \( x^p(t) \) a special type of fundamental solution matrix must be computed. This matrix \( \Phi_T = \Phi(t_0 + T, t_0) \) is called the monodromy matrix and its eigenvalues \( \lambda_i \) are known as Floquet multipliers. This monodromy matrix maps an infinite small perturbation \( w(t_0) \) from a point \( x(t_0) \) on the periodic solution to the response \( w(T) \) at time \( T \) (one period) later.

\[
w(T) = \Phi(t_0 + T, t_0)w(t_0)
\]  

(3.14)

For a autonomous ODE system, one of the Floquet multipliers is always equal to 1. Its eigenvector is the tangent vector to the periodic orbit in the point \( x(t_0) \), so it is equal to \( f(x(t_0)) \). This Floquet multiplier is also called the trivial Floquet multiplier and belongs to an infinite small perturbation in the direction of the local vector field. A periodic solution is asymptotically stable if all Floquet multipliers, except the trivial Floquet multiplier, have a modulus smaller than one.

If the system of collocation equations (3.11) did converge then the residuals \( r_i \) are equal to, or depending on the converge criterium used, very close to zero. With \( r_i \approx 0 \) the equations (3.8) contain a mapping from a perturbation of the point \( x_1 \) on the periodic solution to the corresponding perturbation of the point \( x_{N+1} \) one period later. The matrix \( \Gamma_1 \) maps an infinite small perturbation \( w_1 \) on \( x_1 \) to \( w_2 \) for \( x_2 \) and the matrix \( \Gamma_2 \) maps this resulting perturbation to \( w_3 \). Consequently for the perturbation one period later we can write

\[
w(T) = w_{N+1} = \Gamma_N \cdot \ldots \cdot \Gamma_2 \Gamma_1 w_1.
\]  

(3.15)

Hence the monodromy matrix in \( x_1 \) can be recovered from (3.11) at convergence and is

\[
\Phi_T = \Phi(t_0 + T, t_0) = \Gamma_N \cdot \ldots \cdot \Gamma_2 \Gamma_1.
\]  

(3.16)

In [5] it is stated that this procedure for computation of the monodromy matrix is accurate if \( \varepsilon_{\text{mach}} |\lambda_{\text{max}}| \ll \lambda_{\text{min}} \) with \( \varepsilon_{\text{mach}} \) the machine precision. A more accurate procedure for problems with a very wide range of Floquet multipliers can be found in [5].
Chapter 4

Numerical Results

Now the theoretical background for computing periodic solutions of non-linear autonomous ODEs is made clear we implemented the algorithm in a MATLAB script so we could test its performance. For the quadrature method we applied the method of Gauss so the resulting order of the routine is $2k$ with $k$ the number of collocation points in each interval. For comparison each testcase is also solved with a finite difference routine and a single shooting routine. The finite difference routine uses a trapezoidal scheme, which is of second order. The single shooting routine uses the MATLAB routine ODE45 for the integration. ODE45 is dedicated for solving non-stiff differential equations and is a medium order method. The routine adapts automatically the stepsize to satisfy the user specified tolerance for the error. This error $e(i)$ at a mesh point $x(i)$ is defined by two parameters as $e(i) \leq \max(\text{RelTol} \cdot \text{abs}(x(i)), \text{AbsTol})$.

For the first test case we computed periodic solutions of a non-linear oscillator described by

$$
x_1' = x_1 + \beta x_2 - x_1(x_1^2 + x_2^2)
$$

$$
x_2' = -\beta x_1 + x_2 - x_2(x_1^2 + x_2^2).
$$

This set of non-linear ODEs comes from a very rare class, namely the class of non-linear ODEs for which an analytical expression is known for its limit cycle. The asymptotically stable solution for this set of ODEs is $x_1 = \sin(\beta t)$ and $x_2 = \cos(\beta t)$ which is quite smooth behavior. For $\beta = 1$ the period time $T$ for this system is $2\pi$ and a point on the solution $x_0 = [0, 1]^T$.

Table 4.1 shows the results for the computation of the periodic solution of (4.1) for $\beta = 1$ with the 2 and 4 Gauss points collocation scheme, the second order finite difference scheme and the single shooting method. For each method we used for initial guess $T_0 = 6$ and $x_0 = [0, 0.8]^T$ and as converge criterium $|w| < 10^{-12}$. For the collocation method and the finite difference method we used ODE45 for the integration from $x_0$ for $t = 0..T_0$ to provide the method the initial guess and for both the integration tolerances RelTol and AbsTol we used the value tol.

Note that the finite difference routine used didn't provide the monodromy matrix so for this method no Floquet multipliers are computed.
CHAPTER 4. NUMERICAL RESULTS

<table>
<thead>
<tr>
<th>N / tol</th>
<th>meth.</th>
<th>err. in $T$ [%]</th>
<th>$\lambda_1$</th>
<th>$\lambda_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>cll 2pt</td>
<td>1.6539e-002</td>
<td>9.9997e-001</td>
<td>3.2977e-006</td>
</tr>
<tr>
<td>10</td>
<td>cll 4pt</td>
<td>1.8092e-006</td>
<td>1.0000e+000</td>
<td>3.4873e-006</td>
</tr>
<tr>
<td>10</td>
<td>fd</td>
<td>3.4252e+000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>cll 2pt</td>
<td>1.0559e-003</td>
<td>1.0000e+000</td>
<td>3.4755e-006</td>
</tr>
<tr>
<td>20</td>
<td>cll 4pt</td>
<td>1.4551e-006</td>
<td>1.0000e+000</td>
<td>3.4873e-006</td>
</tr>
<tr>
<td>20</td>
<td>fd</td>
<td>8.3067e-001</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1e-6</td>
<td>sht</td>
<td>2.0799e-004</td>
<td>1.0000e+000</td>
<td>5.6321e-006</td>
</tr>
<tr>
<td>1e-8</td>
<td>sht</td>
<td>9.6051e-007</td>
<td>1.0000e+000</td>
<td>2.5859e-006</td>
</tr>
</tbody>
</table>

Table 4.1: Results for relative error in period time $\left(\frac{\left|T_{\text{comp}}-2\pi\right|}{2\pi}\cdot100\%\right)$ and values for Floquet multipliers $\lambda_1, \lambda_2$ for the non-linear oscillator with $\beta = 1$ ($N =$ number of mesh points, tol = integration tolerance, cll 2pt = 2 point Gauss collocation, cll 4pt= 4 point Gauss collocation, fd= second order finite difference scheme and sht= single shooting method).

From Table 4.1 it can be noted that for the smooth system of (4.1) the collocation method gives accurate results. Given the fact that the number of mesh points $N + 1$ is quite small and in comparison with the results for the other two methods we can therefore conclude that the collocation method performs very well. The differences in the values for the second Floquet multiplier $\lambda_2$ are of order $10^{-6}$ and are of no significance.

From Table 4.1 we can also notice that the error for the 4 point collocation scheme for $N = 10$ is of much smaller order than the error for the 2 point scheme for $N = 20$ given the fact that both computations did use the same number of function evaluations. In Figure 4.1 are, in comparison with the analytical solution for the periodic solution $x_1 = \sin(t)$ and $x_2 = \cos(t)$, the absolute errors $\left|x_\pi(t_i) - x(t_i)\right|$ at the mesh points plotted for these two computations. From Figure 4.1 we can conclude that not only the error for the period time $T$ but also the error at the mesh points is of much less order for the 4 point method. Note that for these computations we didn’t used the orthogonality condition of (3.10) but we fixed the point $x_1(0) = 0$ so that the phase is equal to the analytical expressions.

For the second testcase we used the equation

\[
\begin{align*}
x'_1 &= x_2 \\
x'_2 &= \nu(1 - x_1^2)x_2 - x_1.
\end{align*}
\]  

(4.2)

This equation is introduced by Van der Pol in 1926 as a model for self-sustained oscillations and is known as Van der Pol’s equation.

For this set of ODEs we want to computed the period time $T$ for the values $\nu = 1, 2, \ldots, 10$ by means of sequential continuation. Sequential continuation uses the results from a computation of a periodic solution for a parameter value $\mu$ as an initial guess to compute the periodic solution for a parameter value $\mu + \delta\mu$. The maximum value for
this step in parameter value $\delta \mu$ for which the used method for computation of the periodic solution still converges is a measure for the robustness of the applied method for bad initial guesses. For computation of period solutions for a wide range of parameter values one would like to use an as large as possible stepsize. Therefore the robustness is next to accuracy a performance measurement for comparing different methods.

For the sequential continuation problem we applied for both the collocation- as the finite difference routine $N = 150$ and for the shooting method we used the value $10^{-6}$ for the parameters RelTol and Abstol. For each method we used as converge criterium $|w| < 10^{-12}$ and for startup the initial guess $T_0 = 6.6$ and $x_0 = [2, 0]^T$.

By trial and error we found that, for converge of the sequential continuation for the whole range of $\nu$, the maximum step size to be used is $\delta \nu = 1$ for the collocation method and $\delta \nu = 0.1$ for both the finite difference method as for the shooting method. The observation was that for each decrease of the value for $\delta \nu$ the sequential continuation did get a larger range of converge. For the shooting method this can be explained by the increasing growth $T_{\nu + \delta \nu} - T_{\nu}$ in period time for increasing values of $\nu$. This means that the initial guess for the period time $T_0$ will become worse within each step in $\nu$. The finite difference routine doesn’t use an integration routine when it uses data from the last iteration. The routine uses the $N$ values of $x$ at the mesh points which should be much more robust for an increasing growth in period time. The problem for the robustness of the finite difference routine comes therefore probably from the fact that the trapezoidal scheme is of very low order.

For small values of $\nu$ the Van der Pol’s equation exhibits smooth behavior. But with increasing values of $\nu$ the system becomes less smooth. This behavior is illustrated in figure 4.2 where the periodic solutions plotted for $\nu = 1$, $\nu = 5$ and $\nu = 10$. The more non-smooth the equations become, the harder it becomes for the local approximation,
especially when it is of a low order, to follow the behavior of the ODEs. Consequently the more mesh points are needed for an accurate approximation. Experiments with more mesh points used for the finite difference routine did confirm this. With a stepsize of $\delta \nu = 1$ the sequential continuation did converge with $N = 150$ until $\nu = 6$, with $N = 300$ until $\nu = 8$ and with $N = 400$ for the whole range.

To get insight on the accuracy of the computed solutions we compare the computed period time $T_{comp}$ with accurate computed values $T_{acc}$ from [6]. The results for the relative error in the computed period time are listed in Table 4.2. From Table 4.2 it can be noted that the increasing non-smoothness of the system for increasing values of $\nu$ is reflected by an increasing error for the fixed stepsize methods. After all, the shooting method adapts his stepsize automatically so this wouldn’t be a fair comparison.

If we compare the resulting errors in combination with the used computation time $t^1$ we can conclude that despite the better results for $\nu > 4$ the 4 points collocation method performs significantly better than the other methods. A new experiment with $N = 300$ showed actually errors of the same order as for the shooting method and did require only 138 seconds for computation.

For a fair comparison we did also experiments with other integration routines as ODE15S and ODE23S for implementation in the shooting method. These two routine are both dedicated for stiff differential equations. The implementation of these two routines did result in an increase in computation time, for this sequential continuation ODE45 stays therefore the preferred integration routine for the shooting method.

---

1 All computations are done with MATLAB 6 on a Windows 2000 PC with a Pentium III 500 Mhz processor and 256 Mb RAM.
### Table 4.2: Comparison of relative error in period time $[(T_{\text{comp}} - T_{\text{acc}})/T_{\text{acc}} \cdot 100\%]$ for sequential continuation for $\nu = 1, 2, \ldots, 10$. ($t =$ used cputime, cll 2pts = 2 point Gauss collocation with N=150, cll 4pts= 4 point Gauss collocation with N=150, finite diff.= second order finite difference scheme with N=400 and shooting= single shooting method)

<table>
<thead>
<tr>
<th>$\nu$</th>
<th>Period $T_{\text{acc}}$</th>
<th>cll 2 pts [%]</th>
<th>cll 4 pts [%]</th>
<th>shooting [%]</th>
<th>finite diff. [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.66328686</td>
<td>1.4071e-005</td>
<td>1.9178e-005</td>
<td>1.9167e-005</td>
<td>3.6169e-003</td>
</tr>
<tr>
<td>2</td>
<td>7.6297448</td>
<td>8.0081e-005</td>
<td>1.4432e-005</td>
<td>1.2565e-005</td>
<td>1.4317e-002</td>
</tr>
<tr>
<td>3</td>
<td>8.85909550</td>
<td>7.6580e-004</td>
<td>8.9480e-006</td>
<td>1.9985e-005</td>
<td>4.4291e-002</td>
</tr>
<tr>
<td>4</td>
<td>10.20352369</td>
<td>3.9826e-003</td>
<td>1.2312e-005</td>
<td>2.5246e-005</td>
<td>1.0801e-001</td>
</tr>
<tr>
<td>5</td>
<td>11.61223067</td>
<td>1.5233e-002</td>
<td>1.4719e-005</td>
<td>1.4771e-006</td>
<td>2.2463e-001</td>
</tr>
<tr>
<td>7</td>
<td>14.53974774</td>
<td>1.4064e-001</td>
<td>2.0903e-004</td>
<td>1.9734e-005</td>
<td>7.1519e-001</td>
</tr>
<tr>
<td>8</td>
<td>16.03817623</td>
<td>1.8608e-001</td>
<td>1.6757e-004</td>
<td>1.5192e-005</td>
<td>1.1495e+000</td>
</tr>
<tr>
<td>9</td>
<td>17.55218414</td>
<td>3.9364e-001</td>
<td>2.2696e-003</td>
<td>2.4070e-005</td>
<td>1.7578e+000</td>
</tr>
<tr>
<td>10</td>
<td>19.07836957</td>
<td>1.6114e+000</td>
<td>1.7101e-002</td>
<td>2.1551e-006</td>
<td>2.5818e+000</td>
</tr>
<tr>
<td>$t$</td>
<td>-</td>
<td>23 sec</td>
<td>67 sec</td>
<td>18 min</td>
<td>53 sec</td>
</tr>
</tbody>
</table>

For a last comparison we compared the computed Floquet multipliers for the collocation method and the shooting method. The observation was that for $\nu > 1$ the values for the non trivial multiplier where of great difference. Accurate values for the Floquet multipliers for this range of $\nu$ were not available from literature. For conclusions we examined therefore the value of the non trivial Floquet multiplier from values for $0.1 \leq \nu \leq 3$ with $\delta \nu = 0.1$. The result for this sequential continuation problem, for which the same parameters where used as for the range $1 \leq \nu \leq 10$, is plotted in Figure 4.3.

From the figure we can conclude that the non trivial Floquet multiplier decreases very fast for increasing values of $\nu$. We can observe that for $\lambda < 10^{-5}$ the collocation method, in contrary with the shooting method, gives the expected continuation. For values for $|\lambda|$ near $10^{-15}$ the collocation method exhibits strange continuation, this is probably due the finite machine precision $\varepsilon_{\text{mach}}$ which is for MATLAB of order $10^{-16}$. 
Figure 4.3: Non trivial Floquet multiplier for Van der Pol’s equation computed with shooting method and 4 points Gauss collocation method for $\nu = 0.1, \ldots, 3$ with $\delta \nu = 0.1$.
Chapter 5

Conclusions

With the two numerically testcases we showed that the collocation method for computing periodic solutions is of excellent performance. In the first testcase, the non-linear oscillator, we compared the results for both the solution at the mesh points as well as the period time $T$ with the known solution. From this we could conclude that for this smooth system the collocation method with the application of a Gaussian quadrature scheme is of good accuracy.

For the second testcase we formulated a sequential continuation problem for Van der Pol’s equation. For this problem not only accuracy but also robustness for bad initial guesses is of great importance. For increasing values of $\nu$ we showed that the system become more non-smooth. This resulted in a testcase where both the robustness as the accuracy was tested. By trial and error it was found that for the finite difference method as for the single shooting method the maximum stepsize in $\nu$ for which in each iteration in the sequential continuation problem did converge didn’t exceed 0.1. In comparison with the maximum stepsize for collocation method, which was equal to 1, we could conclude that the collocation method is very robust.

The high order in combination with the robustness of the collocation method resulted in very fast computation for the sequential continuation problem. A drawback for the current implementation of the method of collocation is the need for analytical provided jacobian matrices. Replacement of the analytically provided jacobian matrices with numerical determined, and probably approximated, jacobian matrices will result in an increase in cpu time but this will not affect the accuracy and robustness of the collocation method.
Further Research

Due to the limited timescale for this project the development of the MATLAB implementation of the collocation method for computing periodic solutions did end here. For further research I would like to make two notes.

At the current state the MATLAB script `coll_ps.m` can only work with analytically provided Jacobian matrices. It would be of more convenience if the user doesn’t need to provide analytically Jacobian matrices. Numerically determination of Jacobian matrices is of great expense. It is therefore suggested to work with an approximation for the Jacobian matrix. For approximation of the jacobian matrix at the collocation points one could use a constant jacobian matrix for each subinterval. For the values of the state variables for determine this constant Jacobian matrix one could use perhaps the mean of the state variables at the collocations points so that for each collocation point the error in the Jacobian matrix would be at a minimum. A probably better approximation could follow from the interpolation of the jacobian matrix between jacobian matrices evaluated at the mesh points $x_i$. It should be noted that this approximation for the Jacobian matrices at the collocation points will probably result in a decrease in convergence for the quasi-linearization method.

At last I would like to mention the possibility of a not equally spaced mesh. For the dimensionless form of the BVP the stepsize $h_i$ is decoupled of the period time $T$. For the implementation we used a constant stepsize but this is not required. The implementation of an automatically local mesh selector, which could perhaps use local gradient information, could maybe exploit the high order of method even better. The resulting extra administration of each stepsize in combination with the limited possibilities for sequential continuation should be studied for this implementation.
Bibliography


Appendix A

Matlab Implementation

In this appendix is the MATLAB script for the implementation of the Gauss collocation method for computing periodic solutions is given. Note that this version needs analytical jacobians which are evaluated with the same function file for the ODEs. See also the example in the heading of the script.

```matlab
function [x,t,fi,T,PhiT,conv] =
coll_ps(ffun,x0,T0,fi,diffeps,tol,trace,N,QUAD,ANCH);
% Function coll_ps(ffun,x0,T0,diffeps,tol,trace,N,QUAD,JAC)
% Computes periodic solutions of a set of ODE
% by means of collocation method
% parameters :
% ffun : A function that evaluates the differential equations f(t,x)
% x0 : guess for point or closed orbit [1 x n], fi=[] or [N+1 x n], fi!=[]
% T0 : guess for period time
% fi : derivatives fij on collocation points
% trace= 0 : No output
% 1 : only text
% 2 : text and plot
% N : number of mesh points
% quadrature type:
% QUAD= 1 : 3 point Lobatto
% >2 : QUAD point Gaussian
% ANCH= 1 : f(x1)w1=0
% 2 : x0(1) as fixed point
% Note : to use this option the function ffun needs to
% return the jacobian if it’s called with f(t,x,1)
% Example : function f = vanderpol(t,y,Joption)
% global nu %get value for nu
% if nargin < 3 Joption = 0; end;
% f = [ y(2)
% nu*(1-y(1)^2)*y(2)-y(1) ];
% if Joption == 1
% f = [ 0 1
% end
```

27
APPENDIX A. MATLAB IMPLEMENTATION

%%
%%             -2*nu*y(1)*y(2)-1     nu*(1-y(1)^2) ];
%% end
%% by Niels Mallon, 2002, E-Mail : n.j.mallon@student.tue.nl

if nargin < 5 diffeps = 1e-5; end; if nargin < 6 tol = 1e-6; end;
if nargin < 7 trace = 0; end;
if nargin < 8 N = 100; end; %number of timesteps
if nargin < 9 QUAD = 1; end; %quadrature type
if nargin < 10 ANCH = 1; end; %anchor equation

%setting some variables
n = size(x0,2); %Dimension of the system
I = eye(n); %Identity matrix of size [n,n]
T = T0; %Initial guess for period time
x = []; %variable for state nodes [N,n]
h = 1/N; %time step for mesh (scaled)
t = linspace(0,1,N+1)'; %time nodes (scaled)
Norm=1e99; stop=0; conv=1;

if QUAD==1
    tstr='* Quadrature method : 3 point Lobatto IIIA';
    rho=[0 1/2 1];
    alpha=[0 0 0
           5/24 1/3 -1/24
           1/6 2/3 1/6];
    beta=[1/6 2/3 1/6];
elseif QUAD>1
    tstr=sprintf('* Quadrature method : %d point Gauss-Legendre',QUAD);
    [rho,alpha,beta]=Gquad(QUAD);
end

if trace
    tic
    disp('* Starting collocation method')
    disp(tstr)
end

np=max(size(rho)); %Number of points

D=[]; %D matrix for local parameter elimination
B=[]; %B matrix for determine xij (2.12)
B2=[]; for i=1:np
    D=[D I*beta(i)];
    br=[];
    for j=1:np
        br=[br alpha(i,j)*I];
    end
end
B=[B; br];
B2=[B2; I];
end

Colocation points tij
tij=[]; for i=1:max(size(t))-1
    tij=[tij t(i)+h*rho];
end

if isempty(fi)
    Setting integration options
    The estimated error in each integration step satisfies
    e(i) <= max(RelTol*abs(y(i)),AbsTol)
    options=odeset('RelTol',tol,'AbsTol',tol);

    if trace
        disp('* Integration from x0 for initial guess')
    end
    Integrate system with estimates
    ti = [nsteps,1]
    xi = [nsteps,n]
    [ti,xi] = ode45(ffun,[0,T],x0,options);

    Scale time (tau=t/T)
    ti=ti./T;

    determine values xij for initial guess
    for i=1:n
        xij(:,i) = interp1(ti,xi(:,i),tij,’spline’);
    end
    xij=xij.’;

    setting up the initial mesh xi
    for i=1:n
        x(:,i) = interp1(ti,xi(:,i),t,’spline’);
    end
    x=x.’;

    derivatives fi for initial guess
    fi=[];
    for i=1:max(size(tij))
        fi = [fi; T*feval(ffun,0,xij(:,i),0)]; %derivative
    end

    if trace==2
        figure(1);
        plot(t*T,x); %(1,:),tij*T,x(2,:)
    end
xlabel('t')
title('Initial guess')
end;
else
    x=x0';
end

NM=[]; %empty array for norm
i=0; %Iteration counter
while (Norm>diffeps) & (stop == 0)
i=i+1; %Iteration counter
H = zeros(n*(N+1)+1); %variable for set of equations
R = []; %rhs
FP=[]; %variable for update data for fi
for pp=1:N
    E = zeros(np*(n+1));
    V = [];
    U = [];
    qi = [];
    Q = [];
    s1 = n*np*(pp-1);
    %Determine xij with eq. 2.13
    xij=B2*x(:,pp)+h*B*fi(s1+1:s1+np*n);
    %determine f(xij), df(xij)/dx
    fxi=[];
    dfxi=[];
    for j=1:np
        fxi = [fxi feval(ffun,0,xij((j-1)*n+1:j*n,:),0)]; %derivative
        dfxi = [dfxi feval(ffun,0,xij((j-1)*n+1:j*n,:),1)]; %jacobian
    end
    %Collocation equation
    for qq=1:np %for each coll. point
        s2 = n*(qq-1)+1;
        A = T*dfxi(:,s2:n+s2-1); %A(tij,xij)
        V = [V; A];
        U = [U; fxi(:,qq)];
        qi = [qi; T*fxi(:,qq) - fi(s1+s2:s1+s2+n-1,:)];
        for rr=1:np %column
            s3 = n*(rr-1)+1;
            Q(s2:s2+n-1,s3:s3+n-1) = alpha(qq,rr)*A;
        end
    end
end

%Local parameter elimination
W=eye(np*n)-h*Q;
iW=inv(W);
GAMMA=I+h*D*iW*V;
\[
\text{LAMBDA}=h*D*iW*U;
\]
\[
R=R; \ h*D*iW*qi; \quad \% \text{column with } ri;
\]
\[
\text{FP}=[FP; \ iW*V \ iW*U \ iW*qi]; \quad \% \text{storage for update } fi
\]
\[
\text{rowH}=(pp-1)*n+1:pp*n;
\]
\[
H(\text{rowH},\text{rowH})=-\text{GAMMA};
\]
\[
H(\text{rowH},\text{rowH}+n)=I;
\]
\[
H(\text{rowH},(N+1)*n+1)=-\text{LAMBDA};
\]
\[
\text{end}
\]
\[
H(N*n+1:(N+1)*n,1:n)=I; \quad \% \text{Boundary equations}
\]
\[
H(N*n+1:(N+1)*n,N*n+1:(N+1)*n)=-I;
\]
\[
R=R; \ x(:,N+1)-x(:,1); 0; \quad \% \text{RHS}
\]
\[
\% \text{Anchor equation (orthogonality condition on } x0')
\]
\[
\text{if } \text{ANCH}=1
\]
\[
f1=\text{feval}(ffun,0,x(:,1),0);
\]
\[
H((N+1)*n+1,1:n)=f1';
\]
\[
\text{elseif ANCH}=2
\]
\[
H((N+1)*n+1,1)=1;
\]
\[
\text{end}
\]
\[
\% \text{Create sparse matrix}
\]
\[
H=\text{sparse}(H);
\]
\[
\% \text{Solve linear system}
\]
\[
w = H \backslash R;
\]
\[
\text{wi} = w(1:n*(N+1));
\]
\[
\text{wi} = \text{reshape(wi},n,N+1);
\]
\[
dT = w(n*(N+1)+1);
\]
\[
\% \text{Update variables}
\]
\[
x = x + \text{wi};
\]
\[
T = T + dT;
\]
\[
\text{for } rr=1:N \% \text{Update } fi \text{ with eq. 3.12}
\]
\[
s=np*n*(rr-1);
\]
\[
\text{fi}(s1:s+n*np,:)=\text{fi}(s1:s+n*np,:)+\text{FP}(s1:s+n*np,1:n)*\text{wi}(:,rr)\ldots
\]
\[
+\text{FP}(s1:s+n*np,n+1)*dT+\text{FP}(s1:s+n*np,n+2);
\]
\[
\text{end}
\]
\[
\% \text{Determine } |w| \text{ for converge check}
\]
\[
\text{NormOld} = \text{Norm};
\]
\[
\text{Norm} = \text{norm}(w);
\]
\[
\text{NM}=\text{[NM Norm]};
\]
\[
\text{if trace}
\]
\[
\text{disp(sprintf(''\* Iteration : \%d, T = \%0.7g, Norm = \%0.7g',i,T,\text{Norm}))}
\]
\[
\text{if } i<0
APPENDIX A. MATLAB IMPLEMENTATION

```matlab
figure(1)
subplot(5,1,i+1);
plot(x(:,1),x(:,2));
ylabel(['i = ',num2str(i)]);
v(i,:) = axis;
end

if Norm > NormOld
    if trace
        disp('* not convergent, norm is not decreasing monotonically')
    end
    stop = 1;
    conv = 0;
end

PhiT=I; %Monodromy matrix
for pp=1:N %Determine Monodromy matrix with eq. 3.16
    rowH=(pp-1)*n+1:pp*n;
    GAMMA=-H(rowH,rowH);
    PhiT=GAMMA*PhiT;
end

%rescale t
if trace
    disp(sprintf('* Elapsed time = %0.4g',toc))
end if trace>1
    figure
    semilogy(1:i,NM)
    title(strcat('Convergence ',tstr))
    xlabel('iteration')
    ylabel('norm')
end

function [rho,alpha,beta]=Gquad(n)
%returns scaled Gauss-Legendre quadrature butcher array
%rho = vector with points
%alpha = [nxn] matrix
%beta = vector with weights

%first get the roots and weights
[rts,beta] = GLNodeWt(n);
%Next determine matrix alpha
syms x a=-1; alpha=[];
```
%beta=[];
for i=1:n
  L=1;
  for ii=1:n
    if ii~=i
      L=L*(x-rts(ii))/(rts(i)-rts(ii));
    end
  end
  for iii=1:n
    alpha(iii,i)=int(L,x,a,rts(iii));
  end
  %beta(i)=int(L,x,a,1);
end

%scale integration limits from [-1, 1] to [0, 1]
 rho=(rts'+1)/2; alpha=alpha*0.5; beta=beta*0.5;

%% FROM: http://www.me.pdx.edu/~gerry/nmm/mfiles/byChapter.html ====
function [x,w] = GLNodeWt(n)
% GLNodeWt Nodes and weights for Gauss-Legendre quadrature of arbitrary
% order obtained by solving an eigenvalue problem
% Synopsis: [x,w] = GLNodeWt(n)
% Input: n = order of quadrature rule
% Output: x = vector of nodes
% w = vector of weights
% Algorithm based on ideas from Golub and Welsch, and Gautschi. For a
% condensed presentation see H.R. Schwarz, "Numerical Analysis: A
% Comprehensive Introduction," 1989, Wiley. Original MATLAB
% implementation by H.W. Wilson and L.H. Turcotte, "Advanced Mathematics

beta = (1:n-1)./sqrt(4*(1:n-1).^2 - 1);
J = diag(beta,-1) + diag(beta,1);% eig(J) needs J in full storage
[V,D] = eig(J);
[x,ix] = sort(diag(D));%nodes are eigenvalues, which are on diagonal of D
w = 2*V(1,ix)'.^2; %V(1,ix)' is column vector of first row of sorted V