

A Newton Method For The Continuation Of Invariant Tori

by

Gunjan Singh Thakur

Thesis submitted to the faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of requirements for the degree of

Masters of Science
in
Engineering Science and Mechanics

Dr. Harry Dankowicz
Dr. Scott Hendricks
Dr. Pushkin Kachroo

July 20, 2004
Blacksburg, Virginia

Keywords : Newton method, Invariant tori, Continuation, Dynamical system,
Nonlinear system

Copyright 2004, Gunjan Singh Thakur

A Newton Method For The Continuation Of Invariant Tori

by

Gunjan Singh Thakur

Abstract

This thesis proposes a novel method for locating a p -dimensional invariant torus of an n -dimensional map.

A set of non-linear equations is formulated and solved using the Newton-Raphson scheme. The method requires a set of sampled points on a *guess* invariant torus. An interpolant is passed through these points to compute the pointwise *shift* on the invariant torus, which is used to formulate the equation of invariance for the torus under the given map.

The principal application of this method is to locate invariant tori of continuous systems. These tori occur for continuous dynamical systems having quasiperiodic orbits in state space. The discretization of the continuous system in terms of a map is accomplished in terms of its *flow* function.

Results for one-dimensional invariant tori in two and three-dimensional state space and for two-dimensional invariant tori in three and four-dimensional maps are presented.

Contents

1	Introduction	1
1.1	Dynamical System	1
1.1.1	Invariance	3
1.2	Numerical Analysis	3
1.2.1	Equilibrium points	3
1.2.2	Periodic Orbits	4
1.2.3	Quasiperiodic orbit	8
1.2.4	The Implicit-Function Theorem	9
1.2.5	Continuation	9
2	Interpolation	11
2.1	Introduction	11
2.2	Cubic Spline	12
2.3	Fourier Interpolation	15
2.4	Bicubic Interpolation	17
2.5	Geometric Properties of Curves and Surfaces	18
2.5.1	Properties of a Curve	19
2.5.2	Properties of a Surface	20
3	Continuation of Invariant Tori	22
3.1	One-Dimensional Tori	22

3.2	Two-Dimensional Tori	25
3.3	n -Dimensional Tori	29
4	Results	31
4.1	One-Dimensional Torus	31
4.2	Two-Dimensional Torus	34
4.3	Sweeping along an n -dimensional torus	40
5	Conclusion	43
	Bibliography	45
A	Matlab Code	48
A.1	One-Dimensional Torus	48
A.2	Sweeping Method	59
A.3	Two-dimensional Torus	62
	Vita	87

List of Figures

3.1	The invariant 1-D Tori in 3-D state space	23
3.2	The invariant 2-D Tori in 3-D state space	26
4.1	Continuation of invariant curve of the map (4.1) (a,b,c) in the 2-D cartesian coordinate and (d,e,f) are in the polar coordinate equation (4.3) (taking θ as the parameter) for $\lambda = 2.5$, $\omega = 0.8$ and (a) $\epsilon = 0.1$ (b) $\epsilon = 0.25$ (c) $\epsilon = 0.315$ (d) $\epsilon = 0.1$ (e) $\epsilon = 0.25$ (f) $\epsilon = 0.315$	33
4.2	Continuation of the invariant curve for the flow map of equation (4.5) in 3-D state space for $m = 0.24$ and $m = 0.28$	34
4.3	Continuation of invariant torus in 3-D space for the flow map of equation (4.7) for $\omega = 5.3$, $a = 5$ and (a) $\alpha = 0$, $\beta = 0$ (b) $\alpha = 0$, $\beta = 0.4$ (c) $\alpha = 1.0$, $\beta = 0.4$ (d) $\alpha = 1.5$, $\beta = 0.4$ (e) $\alpha = 1.51$, $\beta = 0.4$ (f) $\alpha = 1.52$, $\beta = 0.4$. . .	35
4.4	Different cross sections of the invariant torus ($\alpha = 1.52$, $\beta = 0.4$, $a = 5.0$, $\omega = 5.3$)	36
4.5	The principal curvature directions and the normal at a point corresponding to $t = 1.0$, $s = 1.0$ on the invariant torus ($\alpha = 1.52$, $\beta = 0.4$, $a = 5.0$, $\omega = 5.3$)	37
4.6	The characteristic topological cylinder for the van-der Pol oscillator for $F = 1$, $\nu = 0.01$, $\omega = 1.01$ and (a) $\epsilon = 0.01$ (b) $\epsilon = 0.015$ (c) $\epsilon = 0.02$ (d) $\epsilon = 0.022$.	38
4.7	The projection of the characteristic topological cylinder in 4-D state space in x_1 - x_2 - x_3 state space for the equation 4.11 obtained by continuation for $\omega = 1.9$, $m = 0.21$ and (a) $F = 0.0$ (b) $F = 0.01$ (c) $F = 0.03$ (d) $F = 0.05$ (e) $F = 0.06$ (f) $F = 0.08$	39
4.8	The characteristic topological cylinder for the van-der Pol oscillator obtained by the sweeping technique, $F = 1$, $\nu = 0.01$, $\epsilon = 0.01$, $\omega = 1.01$	40

4.9	The projection of the characteristic topological cylinder in 4-D state space in x_1 - x_2 - x_3 state space for the equation 4.11 obtained by the sweeping technique, $F = 1, \omega = 1.9, m = 0.21$	41
-----	---	----

Chapter 1

Introduction

1.1 Dynamical System

Any system that evolves in time is a dynamical system. The behaviour of such systems can be chaotic, stochastic, deterministic etc. In this work only those dynamical system which are deterministic are considered. For a given deterministic system, if we know the present state and the law of the evolution of the systems then we can predict the future of the system. Thus the notion of a deterministic dynamical system includes a set of all the possible states (*state space*) and a law for the *evolution* of the state in *time* [1]. If the evolution operator depends only on the time elapsed then the dynamical system is said to be autonomous. If, in addition to elapsed time, the evolution operator depends on absolute time, the system is said to be nonautonomous.

Definition 1 *An autonomous dynamical system is a triple T, X, ϕ^t , where T is a time set, X is a space set, and $\phi^t: X \rightarrow X$ is a family of evolution operators parameterized by $t \in T$ that satisfies the following properties.*

- (1) $\phi^0 = id$ where *id* is the identity map on X
- (2) $\phi^{t+s}x = \phi^t(\phi^s x)$ for all $x \in X$ and $t, s \in T$

The evolution ϕ can either be continuous in time (differential equations) or discrete in time (maps): the corresponding dynamical systems are called *continuous* and *discrete dynamical system*, respectively.

The basic geometrical object in the *state space* of any given dynamical system are its *orbits*. Orbits of a continuous system with a continuous evolution operator are *curves* in state space X parameterized by time t and oriented in the increasing direction of time. Orbits of a

discrete-time system are sequence of points in the state space X enumerated by increasing integers.

In an autonomous dynamical system, there is a unique orbit through any point in state space. In the case of continuous systems, where the evolution operator satisfies a system of ordinary differential equations, this is guaranteed by the existence and uniqueness theorem [14].

The simplest type of orbit is a fixed point [1].

Definition 2 *A point $x_0 \in X$ is called a fixed point (equilibrium point) if $\phi^t x_0 = x_0$ for all $t \in T$ period.*

Consider those continuous dynamical systems that can be written as

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}) \quad (1.1)$$

where $\mathbf{x} \in R^n$ and $\mathbf{F} : R^n \rightarrow R^n$.

Each of the orbits in the state space can be visualized as a flow solution i.e. one can start at some initial guess $\mathbf{x}_0$ and flow along a orbit as the system evolves in time. In this case the evolution operator is called a *flow function*. Equilibrium points correspond to those points in the vector field where $\mathbf{F}(\mathbf{x}) = \mathbf{0}$. Hence they are those points where there is no flow.

Now, consider those discrete dynamical systems that can be written as

$$\mathbf{x}_{m+1} = \mathbf{F}(\mathbf{x}_m) \quad (1.2)$$

where $\mathbf{x} \in R^n$ and $\mathbf{F} : R^n \rightarrow R^n$. A fixed point for such a system is given by $\mathbf{F}(\mathbf{x}^*) = \mathbf{x}^*$, i.e., the point $\mathbf{x}^*$ maps to itself.

Another relatively simple type of orbit is a *periodic orbit* [1].

Definition 3 *A collection of points L_0 in state space is called a periodic orbit if each point $x_0 \in L_0$ satisfies $\phi^{t+T_0} x_0 = \phi^t x_0$ with some $T_0 > 0$, for all $t \in T$.*

The minimal T_0 is called the *period* of the cycle L_0 . For a discrete system any sequence composed of repeated blocks of length N_0 represents a periodic orbit of period N_0 .

For a continuous-in-time dynamical system, periodic orbit corresponds to a Fourier expansion of the state as a function of time with one fundamental frequency. In contrast, a quasiperiodic orbit corresponds to a Fourier expansion of the state as a function of time with multiple incommensurate fundamental frequencies. A quasiperiodic orbit can be shown to lie on a torus in state space.

Definition 4 *A p -dimensional torus T^p is the image under a smooth map of all ordered sequences having p components $\in S^1 \times S^1 \dots \times S^1$ with no singular point.*

An example of a two-dimensional torus is given by the regular donut-shaped surface in three-dimensional space R^3 .

1.1.1 Invariance

The concept of invariance is very important in dynamics. Many simple, invertible, differentiable dynamical systems can have very complex closed invariant sets.

Definition 5 *An invariant set of a dynamical system $\{T, X, \phi^t\}$ is a subset $S \subset X$ such that $x_0 \in S$ implies $\phi^t x_0 \in S$ for all $t \in T$.*

From this definition it is clear that any collection of orbits is invariant.

1.2 Numerical Analysis

In this section we look at different numerical techniques that are used to compute the basic types of invariant sets for a given dynamical system. Once these invariant sets are computed for a given set of system parameters one can perform a continuation of these sets under variations in system parameters to study how the invariant sets change shape and bifurcate.

1.2.1 Equilibrium points

The problem of finding an equilibrium point for both a continuous system (1.1) and a discrete dynamical system (1.2) reduces to finding the solution for an equation of the form $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ where $\mathbf{f}(\mathbf{x}) = \mathbf{F}(\mathbf{x})$ for a continuous system and $\mathbf{f}(\mathbf{x}) = \mathbf{F}(\mathbf{x}) - \mathbf{x}$ for the discrete system. The Newton-Raphson method is the most widely used method to find roots of a given function. In the following discussion this method is described.

Newton-Raphson

Given

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{1.3}$$

one wishes to find the solution of the above equation. Let $\mathbf{x} = \mathbf{x}_0$ be an initial guess and suppose that $\mathbf{x}_0 + \Delta\mathbf{x}$ is one of the solutions of the above equation. The Taylor expansion of $\mathbf{f}(\mathbf{x})$ about the point $\mathbf{x}_0$ can be written as

$$\mathbf{f}(\mathbf{x}_0 + \Delta\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + \mathbf{f}_{\mathbf{x}}(\mathbf{x}_0) \Delta\mathbf{x} + \mathcal{O}(2) + \dots$$

Suppose that the Jacobian matrix $\mathbf{f}_{\mathbf{x}}(\mathbf{x}_0)$ is invertible. Then, neglecting terms of order two or higher yields

$$\Delta\mathbf{x} \approx -(\mathbf{f}_{\mathbf{x}}(\mathbf{x}_0))^{-1} \cdot \mathbf{f}(\mathbf{x}_0).$$

Thus, a better approximation of the equilibrium point should be

$$\mathbf{x}_0 - (\mathbf{f}_{\mathbf{x}}(\mathbf{x}_0))^{-1} \cdot \mathbf{f}(\mathbf{x}_0)$$

Repeating this process, we arrive at the iterative formula

$$\mathbf{x}_{i+1} = \mathbf{x}_i - (\mathbf{f}_{\mathbf{x}}(\mathbf{x}_i))^{-1} \cdot \mathbf{f}(\mathbf{x}_i) \quad (1.4)$$

that can be shown to converge quadratically fast to the solution of equation 1.3, provided that the initial guess is sufficiently good. The convergence is independent of the stability of the equilibrium point under the evolution operator, i.e., the local behavior of nearby orbits in state space.

The most computationally expensive step in Newton algorithm is the calculation of the inverse of the Jacobian. They may be somewhat overcome by solving the following equation

$$0 = (\mathbf{f}_{\mathbf{x}}(\mathbf{x}_i))\eta_i = -\mathbf{f}(\mathbf{x}_i) \quad (1.5)$$

such that $\eta_i = -(\mathbf{f}_{\mathbf{x}}(\mathbf{x}_i))^{-1}\mathbf{f}(\mathbf{x}_i)$, instead of first computing the inverse and then performing the multiplication. In another approach, one can also reduce the computational effort by computing the Jacobian just once at the beginning of an iteration and using the calculated value for all subsequent iterations. If the initial guess is close to the root of the problem, convergence is again assured but is only linearly fast. This method is referred to as the *Newton-chord* method. A few other modifications aimed to reduce the computational cost can be found in [1, 17].

1.2.2 Periodic Orbits

This section gives a brief overview of different numerical methods used to compute periodic orbits for a given continuous system. The simplest way to compute a periodic orbit is by direct numerical simulation, i.e. one chooses an initial condition and integrates the system to converge on the periodic orbit. Typically, this approach only works for an attracting periodic orbit, i.e., one for which nearby orbits in state space approach the periodic orbit asymptotically in forward time. Even for attracting periodic orbits, it may take a long time to converge for systems with small amount of damping. Furthermore, in the presence of multiple system attractors, there is no guarantee that the integration will converge to the desired orbit. To overcome these problems many approaches in the frequency domain and time domain have been proposed.

The frequency-domain approach leads to various spectral methods [18, 19, 20]. These methods are not considered in detail in this thesis. The time-domain formulation leads to methods such as the shooting method [23], finite-difference schemes [21, 22], the collocation method [25], Poincaré [22, 24] map methods, and others. The following discussion gives a brief overview of these commonly used techniques.

For a general periodic orbit, assume that the position of the orbit is known approximately and one wishes to seek a more accurate position of at least one of the points on the orbit. The period of the orbit T_0 is also unknown in an autonomous system. It is convenient to formulate the problem as a boundary value problem over a fixed interval. Introducing a transformation $t = T_0\tau$, equation 1.1 can be written as

$$\frac{d\mathbf{x}}{d\tau} = T_0\mathbf{F}(\mathbf{x}) \quad (1.6)$$

Then, if $\mathbf{x}(\tau)$ corresponds to a periodic orbit with period T_0 , it follows that

$$\mathbf{x}(0) = \mathbf{x}(1) \quad (1.7)$$

The periodicity condition given by equation 1.7 does not uniquely define the periodic solution. An extra *phase condition* is needed in order to select a solution from all the possible solutions, for example

$$\psi(\mathbf{x}_0) = 0 \quad (1.8)$$

where $\psi : R^N \rightarrow R^1$ is a scalar function defined on the periodic solution. Thus equations 1.6, 1.7 and 1.8 completely defines the boundary value problem.

The most commonly used scalar functions for ψ are as follows

1. A hypersurface $g(\mathbf{x})$ in R^N which passes through $\mathbf{x}(0)$.

$$\psi(\mathbf{x}) = g(\mathbf{x}) = 0 \quad (1.9)$$

some known reference curve $\mathbf{v}(\tau)$ with period one

$$\psi(\mathbf{x}) = \langle \mathbf{x} - \mathbf{v}(0), \dot{\mathbf{v}}(0) \rangle = 0 \quad (1.10)$$

2. The most commonly used condition called the *integral phase condition*. Consider some known reference curve $\mathbf{v}(\tau)$ with period one

$$\psi(\mathbf{x}) = \int_0^1 \langle \mathbf{x}(\tau), \dot{\mathbf{v}}(\tau) \rangle d\tau = 0 \quad (1.11)$$

Shooting Method

In this method we formulate the boundary value problem using 1.6, 1.7 and 1.9. Let $\phi(\mathbf{x}, \tau)$ be the solution of equation 1.6, It follows that

$$\begin{aligned} \phi(\mathbf{x}(0), T_0) - \mathbf{x}(0) &= \mathbf{0} \\ g(\mathbf{x}(0)) &= 0 \end{aligned} \quad (1.12)$$

The above $(n + 1)$ -dimensional system can be solved for $n + 1$ unknowns, for example, using the Newton-Raphson scheme with some good initial guess T_0 and $\mathbf{x}(0)$. The Jacobian

matrix corresponding to equation 1.12 can be obtained by numerical differentiation or by the variational equations corresponding to equation 1.6.

Finite Difference

In this method the derivative in equation 1.6 is replaced by a central difference approximation. i.e. one discretizes the time domain by a sufficient large number of mesh points.

$$\frac{d\mathbf{x}(\tau_i)}{d\tau} \approx \frac{\mathbf{x}_{i+1} - \mathbf{x}_{i-1}}{\tau_{i+1} - \tau_{i-1}} \quad (1.13)$$

where $\mathbf{x}_i = \mathbf{x}(\tau_i)$, $i = 1, 2, \dots, N-1$ and $\mathbf{x}_0 = \mathbf{x}(0)$, $\mathbf{x}_N = \mathbf{x}(1)$. Thus, formulating the problem again as a boundary value problem using 1.6, 1.13, 1.7 and 1.9

$$\begin{aligned} \mathbf{x}_{i+1} - \mathbf{x}(i-1) - (\tau_{i+1} - \tau_{i-1})T_0\mathbf{F}\left(\frac{1}{2}(\mathbf{x}_{i+1} + \mathbf{x}_i)\right) &= \mathbf{0} \\ \mathbf{x}_N - \mathbf{x}_0 &= \mathbf{0} \\ \psi(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_N) &= 0 \end{aligned} \quad (1.14)$$

The above $(nN+1)$ -dimensional system can again be solved for the $nN+1$ unknowns as in the previous case. The Jacobian corresponding to the above system can be shown to have a banded structure which reduces the computational cost of the required inversion.

Collocation Method

In this method the boundary value problem is formulated using equations 1.6, 1.7 and 1.11. The time domain is discretized in sufficiently large mesh points

$$0 = \tau_0 < \tau_1 < \tau_2 \dots < \tau_N = 1 \quad (1.15)$$

Then m more points are introduced between two consecutive grid points i.e. between τ_j and τ_{j+1} .

$$\tau_j < \tau_{j,1} < \tau_{j,2} < \tau_{j,3} \dots < \tau_{j,m} < \tau_{j+1}$$

One then seeks a piecewise differentiable and continuous polynomial composed of polynomial functions of at most degree m in each interval $[\tau_i, \tau_{i+1}]$. This interpolating polynomial should satisfy equation 1.6 at each of the m points for every interval. i.e.

$$\left. \frac{d\mathbf{x}}{d\tau} \right|_{\tau=\tau_{j,i}} = T_0\mathbf{F}(\mathbf{x}(\tau_{j,i}))$$

where $i = 1, 2, \dots, m$ and $j = 0, 1, \dots, N-1$. Equation 1.7 takes the form

$$\mathbf{x}_{0,0} = \mathbf{x}_{N-1,m}$$

and equation 1.11 takes the form

$$\sum_{j=0}^{N-1} \sum_{i=0}^m \omega_j < \mathbf{x}_{j,i}, \dot{\mathbf{v}}_{j,i} > = 0$$

where ω_j is the coefficient vector for the j^{th} polynomial and $\mathbf{v}_{j,i}$ is the value of the reference periodic function at $\tau_{j,i}$. The above $(nN+1)$ dimensional system can again be solved for the $nN+1$ unknowns as in the previous case [1].

Poincaré Method

Periodic orbits are most conveniently studied through the introduction of a Poincaré section and its associated Poincaré map. Suppose that the periodic orbit intersects the zero-level surface of the function $g(\mathbf{x})$ transversally at a point $\mathbf{x}^*$, i.e.,

$$g(\phi(\mathbf{x}^*, T)) = g(\mathbf{x}^*) = 0 \quad (1.16)$$

and

$$g_{\mathbf{x}}(\mathbf{x}^*) \cdot \mathbf{F}(\mathbf{x}^*) \neq 0. \quad (1.17)$$

Now consider the scalar-valued function

$$h(\mathbf{x}, t) = g(\phi(\mathbf{x}, t)). \quad (1.18)$$

Then,

$$h(\mathbf{x}^*, T) = 0 \quad (1.19)$$

and

$$\begin{aligned} h_t(\mathbf{x}^*, T) &= g_{\mathbf{x}}(\phi(\mathbf{x}^*, T)) \cdot \frac{\partial}{\partial t} \phi(\mathbf{x}^*, T) \\ &= g_{\mathbf{x}}(\mathbf{x}^*) \cdot \mathbf{F}(\phi(\mathbf{x}^*, T)) \\ &= g_{\mathbf{x}}(\mathbf{x}^*) \cdot \mathbf{F}(\mathbf{x}^*) \neq 0. \end{aligned} \quad (1.20)$$

The implicit function theorem then implies the existence of a unique function $t(\mathbf{x})$ for $\mathbf{x} \approx \mathbf{x}^*$, such that

$$t(\mathbf{x}^*) = T, \quad (1.21)$$

$$h(\mathbf{x}, t(\mathbf{x})) = g(\phi(\mathbf{x}, t(\mathbf{x}))) = 0 \quad (1.22)$$

and

$$\begin{aligned} t_{\mathbf{x}}(\mathbf{x}^*) &= -(h_t(\mathbf{x}^*, T))^{-1} \cdot h_{\mathbf{x}}(\mathbf{x}^*, T) \\ &= -\frac{g_{\mathbf{x}}(\mathbf{x}^*)}{g_{\mathbf{x}}(\mathbf{x}^*) \cdot \mathbf{F}(\mathbf{x}^*)} \phi_{\mathbf{x}}(\mathbf{x}^*, T). \end{aligned} \quad (1.23)$$

This shows that for each trajectory based at an initial point near $\mathbf{x}^*$ there exists a unique elapsed time near T until the trajectory intersects the Poincaré section corresponding to $g(\mathbf{x}) = 0$. Thus, we can define a Poincaré map as

$$\mathbf{P}(\mathbf{x}) = \phi(\mathbf{x}, t(\mathbf{x})), \quad (1.24)$$

that maps points near $\mathbf{x}^*$ onto the Poincaré section. Specifically,

$$\mathbf{P}(\mathbf{x}^*) = \mathbf{x}^* \quad (1.25)$$

is a fixed point of the Poincaré map, i.e. the point x^* on the periodic orbit is a fixed point for $\mathbf{P}$.

1.2.3 Quasiperiodic orbit

In this section we look briefly at some of the techniques used to compute two-dimensional invariant tori [30]. Most of these methods have been proposed in the last twenty years. There are two distinct approaches to solving this problem. One of them attempts to evaluate the quasi-periodic orbit on the torus, while the other attempts to evaluate the torus itself.

The idea of using the spectral balance method to approximate the quasi-periodic orbit is very common. This method is the generalization of the harmonic balance method [27, 28, 22]. Another method uses the idea of computing the quasi-periodic orbit as a fix point to a generalized Poincaré map [2, 3, 4]. However, these methods have one drawback, namely the small divisor problem. The small divisor problem refers to those quasi-periodic orbits for which the rotation number can be approximated by a continued expansion fraction, it becomes hard or even impossible to compute the generalized Poincaré map with sufficient precision. Many of these methods can be extended to orbits in higher dimensions.

Many different techniques are available for computing the two-dimensional torus itself. The basic idea of this approach is to find a torus function that is invariant under the map or the vector field of a given dynamical system [26, 29]. The approach using maps results in applying the Newton-Raphson method to a functional equation whereas the approach using the vector field leads to the application of the Newton-Raphson method to the graph transforms. The method discussed in this thesis falls in the first category, i.e. invariant tori of maps.

The basic idea is to find a torus function $u : T^p \rightarrow R^n$ (T^p is the p-dimensional torus) such that its image $K: u(\theta) | \theta \in T^p$ is invariant under the map $\mathbf{F} : R^n \rightarrow R^n$. That is, the invariance condition

$$u(\mathcal{T}(\theta)) = \mathbf{F}(u(\theta)) \quad (1.26)$$

holds point-wise, where $\theta \in T^p$ and $\mathcal{T} : T^p \rightarrow T^p$ restricts the map $\mathbf{F}$ to the invariant torus K . The above equation provides only an equation in u , but the function $\mathcal{T}$ is also unknown and its depends on the parameterization of the torus K . Once $\mathcal{T}$ is fixed one can solve for the torus. In case the function $\mathbf{F}$ is known in the following form

$$\mathbf{F}(\theta, r) = (g(\theta, r), f(\theta, r)) \quad (1.27)$$

where $F : S^1 \times R \rightarrow S^1 \times R$, $F : S^1 \times R \rightarrow S^1$, $F : S^1 \times R \rightarrow R$.

Assuming that there exist a invariant curve and the r component can be parametrized by θ . Then the invariant condition takes the following functional form for the r component of the equation[5].

$$u(f(u(\theta), \theta)) = g(u(\theta), \theta) \quad (1.28)$$

In this work, a method is proposed for the calculation of the torus of a map by defining the function $\mathcal{T}$ for any general N-dimensional problem (chapter[]). The formulation proposed reduces to the functional form given by equation 1.28, if we have the dynamical system is of the form given be the equation 1.27.

1.2.4 The Implicit-Function Theorem

Suppose that $\mathbf{F}$ is a continuously differentiable vector valued function on some neighborhood W of a point (x_0, μ_0) , such that

$$\mathbf{F}(x_0, \mu_0) = \mathbf{0} \quad (1.29)$$

and assume that $[\mathbf{F}_x(x_0, \mu_0)]^{-1}$ exists, then there exists a smooth locally defined function $\mathbf{x} = \mathbf{f}(\mu)$, such that

$$\mathbf{f}(\mu_0) = \mu_0 \quad (1.30)$$

$$\mathbf{F}(\mathbf{f}(\mu), \mu) = 0 \quad (1.31)$$

for all μ in neighborhood of (μ_0) . Moreover

$$\mathbf{f}_\mu(\mu_0) = -[\mathbf{F}_x(x_0, \mu_0)]^{-1} \mathbf{F}_\mu(x_0, \mu_0) \quad (1.32)$$

1.2.5 Continuation

This section looks at the continuation of invariant sets under variations in as one of the system parameters. Most of the continuation algorithms implement three basic steps

1. Prediction.
2. Correction.
3. Step size.

In the prediction step, one extrapolates information about the solution for $\lambda = \lambda_0$ to obtain an approximation of the solution for $\lambda = \lambda_0 + \delta\lambda$. This predicted solution is taken as a guess solution for some corrective scheme, which is usually a Newton-Raphson based scheme. The step size i.e. $\delta\lambda$ may be continuously varied according to the convergence or lack thereof of the Newton iterations.

The continuation method in which the predicted solution for $\lambda_0 + \delta\lambda$ is taken as the solution from the last converged solution i.e. for λ_0 is called as the *sequential continuation method*. This is the most common method used, especially for tori.

The general continuation method for equilibria and periodic cycles requires to find a functional relationship between the system parameter λ and $\mathbf{x}$ (equilibria point or a point on the periodic orbit). This functional relationship is essentially a curve M in R^{n+1} space. One of the prediction schemes is based on equation 1.32.

In another method, *arclength continuation*, the curve M is parameterized by the arc length l , i.e., $\mathbf{x} = \mathbf{x}(l)$ and $\lambda = \lambda(l)$. Thus, one seeks

$$\mathbf{F}(\mathbf{x}(l), \lambda(l)) = \mathbf{0} \quad (1.33)$$

Differentiating w.r.t l gives n scalar equations

$$\mathbf{F}_{\mathbf{x}}(\mathbf{x}, \lambda)\mathbf{x}' + \mathbf{F}_{\lambda}(\mathbf{x}, \lambda)\lambda' = \mathbf{0} \quad (1.34)$$

Since there are $n+1$ unknowns we need another equation to solve for the tangent vector $[\mathbf{x}' \lambda']$ at $(\mathbf{x}, \lambda)$ on M . This additional condition is obtained by normalizing the Euclidian norm.

$$\mathbf{x}'^T \mathbf{x}' + \lambda'^2 = 1 \quad (1.35)$$

This prediction method is called *tangent prediction* [7].

Chapter 2

Interpolation

In this chapter, the problem of interpolation of a given set of sampled data points is discussed. Polynomial and trigonometric functions are most commonly used to fit a smooth function passing through the given data set. Sections 2.2 and 2.3 describe the concept of interpolation of curves using cubic splines and Fourier series, respectively. The idea of cubic splines is subsequently extended to the two-dimensional interpolation of spatial data (see section 2.4). The chapter concludes with a discussion of the use of the obtained interpolated curve and surface for the computation of certain geometric properties such as the radius of curvature at a given point of a curve and the principal curvatures and the principal directions of curvature at a given point on a surface.

The material in this section serves to support the development of a finite-dimensional Newton method for the continuation of invariant tori of maps in the next chapter.

2.1 Introduction

Consider the N -dimensional space $\mathbb{R}^N$ and let

$$\mathbf{y} = (y^1, y^2, \dots, y^N) \quad (2.1)$$

denote the standard coordinates of an arbitrary point. A hypersurface in this space can be represented as

$$\mathbf{y} = \mathbf{y}(\mathbf{x}) \quad (2.2)$$

or

$$y^1 = y^1(\mathbf{x}), \quad y^2 = y^2(\mathbf{x}), \quad \dots, \quad y^N = y^N(\mathbf{x}) \quad (2.3)$$

where the independent variable vector $\mathbf{x}$ is given as

$$\mathbf{x} = (x_1, x_2, \dots, x_r) \quad (2.4)$$

The representation of a hypersurface by equation 2.2 is known as its *parametric representation*. For $r = 1$ and $r = 2$, equation 2.2 represents a curve or a two-dimensional surface, respectively, in $\mathbb{R}^N$.

In many applications in engineering and science it is important to be able to interpolate discretely sampled point $\mathbf{y}_j$, $j = 0, \dots, M$ of some function on a grid of values of the independent variables $\mathbf{x}_j$, in order to compute the function's value at intermediate points in the domain of definition. The task is to find an appropriate *interpolating function* that passes through the given data points, while satisfying appropriate conditions on its functional characteristics, e.g., smoothness and periodicity.

Since the coordinate $y^1, y^2, \dots, y^N$ are independent, one can find an interpolant function $F^\alpha(\mathbf{x})$ for each coordinate independent of the interpolant for the other coordinates. The most commonly used functions for this purpose are expressions in terms of polynomial and trigonometric functions.

The use of trigonometric function leads to formulations based on Fourier analysis. These formulations typically require that data are sampled at uniformly distributed grid values of the independent variable $\mathbf{x}$. A Fourier interpolant thus obtained is infinitely many times differentiable.

One approach to interpolation with polynomials requires that an n^{th} -degree polynomial be found that passes through the $n + 1$ data points. In contrast to the Fourier interpolant, these sample points need not be uniformly sampled in the independent variable $\mathbf{x}$. However, the interpolated polynomial tends to oscillate between the nodes. To overcome this problem we define the interpolating polynomial as a combination of polynomials of some lower degree with one such polynomial for each interval between two consecutive sampled points. Depending on the degree of the polynomial used, one can impose smoothness of the interpolated curve over the domain. This interpolated curve obtained is called as *spline*.

2.2 Cubic Spline

In this section, we consider the derivation of the functional form for a cubic spline for a curve in $\mathbb{R}^N$, i.e., one independent variable x . The spline $F^\alpha(x)$ is a combination of piecewise defined cubic polynomials that interpolates the given data. Specifically, the spline is constructed to ensure $\mathcal{C}^2$ continuity at each interior data point. Given sampled data (x_k, y_k^α) , $k = 0, \dots, M$, let

$$\Delta : a = x_0 < x_1 < \dots < x_M = b \quad (2.5)$$

be the given partition of the interval $[a, b]$. A function $F^\alpha(x)$ defined on $[a, b]$ is a cubic spline associated with the partition Δ if

1. $F^\alpha(x) \in \mathcal{C}^2[a, b]$

2. $F^\alpha(x)$ coincides with a cubic polynomial on the every subinterval $[x_{k-1}, x_k]$ for $k = 1, \dots, M$
3. $F^\alpha(x_k) = y_k^\alpha$ for $k = 0, \dots, M$

Two additional boundary conditions are necessary to uniquely define a cubic spline, for example,

1. $(F^\alpha)''(x_0) = (F^\alpha)''(x_M) = 0$ (natural spline)
2. $(F^\alpha)'(x_0) = (F^\alpha)'(x_M), (F^\alpha)''(x_0) = (F^\alpha)''(x_M)$ (periodic spline, assuming $y_0^\alpha = y_M^\alpha$)
3. $(F^\alpha)'''(x_1^-) = (F^\alpha)'''(x_1^+), (F^\alpha)'''(x_{M-1}^-) = (F^\alpha)'''(x_{M-1}^+)$ (not-a-knot)

The main objective of this thesis is the continuation of tori, i.e., hypersurfaces that can be parametrized using functions periodic in the independent variables. Thus, in the discussion that follows, we restrict attention to the case of a periodic spline and the associated boundary conditions. The spline $F^\alpha(x)$ can be represented as

$$F^\alpha(x) = y_{k-1}^\alpha p_1(t(x)) + y_k^\alpha p_2(t(x)) + h_k [m_{k-1}^\alpha q_1(t(x)) + m_k^\alpha q_2(t(x))] \quad (2.6)$$

for $x_{k-1} \leq x \leq x_k$, where

$$\begin{aligned} p_1(t) &= (t-1)^2(1+2t) \\ p_2(t) &= t^2(3-2t) \\ q_1(t) &= t(t-1)^2 \\ q_2(t) &= t^2(t-1) \\ t(x) &= \frac{x - x_{k-1}}{h_k} \end{aligned}$$

and

$$h_k = x_k - x_{k-1}$$

It is straightforward to show that

$$m_k^\alpha = (F^\alpha)'(x_k) \quad (2.7)$$

and, therefore, that the function $F^\alpha(x)$ is $\mathcal{C}^1$ on the interior sample points. Indeed, if we let $m_0^\alpha = m_M^\alpha$ (assuming that $y_0^\alpha = y_M^\alpha$), F^α can be extended to a periodic $\mathcal{C}^1$ function for all x . The second derivative of $F^\alpha(x)$ is given as

$$(F^\alpha)''(x) = y_{k-1}^\alpha \frac{12t-6}{h_k^2} + y_k^\alpha \frac{6-12t}{h_k^2} + m_{k-1}^\alpha \frac{6t-4}{h_k} + m_k^\alpha \frac{6t-2}{h_k} \quad (2.8)$$

We proceed to impose the $\mathcal{C}^2$ continuity condition across the internal nodes. Specifically,

$$\lim_{x \rightarrow x_k^-} (F^\alpha)''(x) = y_k^\alpha \frac{-6}{h_k^2} + y_{k-1}^\alpha \frac{6}{h_k^2} + m_{k-1}^\alpha \frac{2}{h_k} + m_k^\alpha \frac{4}{h_k} \quad (2.9)$$

and

$$\lim_{x \rightarrow x_k^+} (F^\alpha)''(x) = y_k^\alpha \frac{6}{h_k^2} + y_{k-1}^\alpha \frac{-6}{h_k^2} + m_{k-1}^\alpha \frac{-4}{h_k} + m_k^\alpha \frac{-2}{h_k} \quad (2.10)$$

$\mathcal{C}^2$ continuity at the k^{th} internal node (i.e., where $k = 1, \dots, M-1$) implies

$$y_k^\alpha \frac{-6}{h_k^2} + y_{k-1}^\alpha \frac{6}{h_k^2} + m_{k-1}^\alpha \frac{2}{h_k} + m_k^\alpha \frac{4}{h_k} = y_{k+1}^\alpha \frac{6}{h_{k+1}^2} + y_k^\alpha \frac{-6}{h_{k+1}^2} + m_k^\alpha \frac{-4}{h_{k+1}} + m_{k+1}^\alpha \frac{-2}{h_{k+1}} \quad (2.11)$$

Rearranging the terms, we have

$$m_{k-1}^\alpha \frac{2}{h_k} + m_k^\alpha \left[\frac{4}{h_k} + \frac{4}{h_{k+1}} \right] + m_{k+1}^\alpha \frac{2}{h_{k+1}} = \frac{6}{h_k^2} (y_k^\alpha - y_{k-1}^\alpha) + \frac{6}{h_{k+1}^2} (y_{k+1}^\alpha - y_k^\alpha) \quad (2.12)$$

Imposing continuity of the second derivative at the M^{th} node (recall that $y_0^\alpha = y_M^\alpha$ and $m_0^\alpha = m_M^\alpha$) yields

$$y_{M-1}^\alpha \frac{6}{h_M^2} + y_M^\alpha \frac{-6}{h_M^2} + m_{M-1}^\alpha \frac{2}{h_M} + m_M^\alpha \frac{4}{h_M} = y_M^\alpha \frac{-6}{h_1^2} + y_1^\alpha \frac{6}{h_1^2} + m_M^\alpha \frac{-4}{h_1} + m_1^\alpha \frac{-2}{h_1} \quad (2.13)$$

or

$$m_{M-1}^\alpha \frac{2}{h_M} + m_M^\alpha \left[\frac{4}{h_M} + \frac{4}{h_1} \right] + m_1^\alpha \frac{2}{h_1} = \frac{6}{h_1^2} (y_1^\alpha - y_M^\alpha) + \frac{6}{h_M^2} (y_M^\alpha - y_{M-1}^\alpha) \quad (2.14)$$

If we let

$$\begin{aligned} \lambda_k &= \frac{1}{h_k} \\ \mu_k &= 4(\lambda_k + \lambda_{k+1}) \\ d_k^\alpha &= 6\lambda_k^2 (y_k^\alpha - y_{k-1}^\alpha) + 6\lambda_{k+1}^2 (y_{k+1}^\alpha - y_k^\alpha) \end{aligned}$$

for $k = 1, \dots, M$, the resultant equations can be written as

$$\begin{pmatrix} \mu_1 & 2\lambda_2 & 0 & 0 & \cdots & 0 & 0 & 2\lambda_1 \\ 2\lambda_2 & \mu_2 & 2\lambda_3 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 2\lambda_3 & \mu_3 & 2\lambda_4 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 2\lambda_{M-1} & \mu_{M-1} & 2\lambda_M \\ 2\lambda_1 & 0 & 0 & 0 & \cdots & 0 & 2\lambda_M & \mu_M \end{pmatrix} \begin{pmatrix} m_1^\alpha \\ m_2^\alpha \\ m_3^\alpha \\ \vdots \\ m_{M-1}^\alpha \\ m_M^\alpha \end{pmatrix} = \begin{pmatrix} d_1^\alpha \\ d_2^\alpha \\ d_3^\alpha \\ \vdots \\ d_{M-1}^\alpha \\ d_M^\alpha \end{pmatrix} \quad (2.15)$$

or

$$\mathbf{A}(x_1, \dots, x_M) \mathbf{M}^\alpha(x_1, \dots, x_M, y_1, \dots, y_M) = \mathbf{D}^\alpha(x_1, \dots, x_M, y_1, \dots, y_M) \quad (2.16)$$

The above equation can be solved to yield the slope m_k^α at each node, which when substituted in equation(2.6) gives the cubic spline.

In the formulation of the Newton method for continuation of invariant tori, it is necessary to compute the derivative of the interpolant with respect to the sampled data values. Specifically, differentiating (2.15) w.r.t $(y_1, \dots, y_M)$ yields

$$\frac{\partial \mathbf{M}^\alpha}{\partial (y_1, \dots, y_M)}(x_1, \dots, x_M, y_1, \dots, y_M) = \mathbf{A}^{-1}(x_1, \dots, x_M) \frac{\partial \mathbf{D}^\alpha(x_1, \dots, x_M, y_1, \dots, y_M)}{\partial (y_1, \dots, y_M)} \quad (2.17)$$

where,

$$\begin{aligned} & \frac{\partial \mathbf{D}^\alpha(x_1, \dots, x_M, y_1, \dots, y_M)}{\partial (y_1, \dots, y_M)} \\ &= \begin{pmatrix} 6(\lambda_1^2 - \lambda_2^2) & 6\lambda_2^2 & 0 & \dots & 0 & 0 & -6\lambda_1^2 \\ -6\lambda_2^2 & 6(\lambda_2^2 - \lambda_3^2) & 6\lambda_3^2 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -6\lambda_{M-1}^2 & 6(\lambda_{M-1}^2 - \lambda_M^2) & 6\lambda_M^2 \\ 6\lambda_1^2 & 0 & 0 & \dots & 0 & -6\lambda_M^2 & 6(\lambda_M^2 - \lambda_1^2) \end{pmatrix} \end{aligned}$$

2.3 Fourier Interpolation

In this section, we consider the derivation of the an interpolant $F^\alpha(x)$ using trigonometric functions for a curve in $\mathbb{R}^N$, i.e., one independent variable x . Given sampled data (x_k, y_k^α) , such that $x_k = kh$, $k = 1, \dots, M$ and M is *even*, the corresponding Fourier interpolant $F^\alpha(x)$ is given by

$$F^\alpha(x) = \frac{1}{M} \left[\sum_{n=-M/2+1}^{M/2-1} H_n^\alpha \exp\left(-2\pi i \frac{nx}{Mh}\right) + H_{M/2}^\alpha \cos\left(-\pi \frac{x}{h}\right) \right] \quad (2.18)$$

where for arbitrary n

$$H_n^\alpha = \sum_{l=1}^M y_l^\alpha \exp\left(2\pi i \frac{nl}{M}\right) \quad (2.19)$$

The collection of the coefficients H_n^α is commonly known as the *discrete Fourier transform* of the sampled data.

The interpolant is a finite trigonometric polynomial with frequencies that are integer multiples of the fundamental frequency $\frac{1}{Mh}$. The largest frequency that appears in the interpolant is called the Nyquist frequency ($f_{\text{Nyquist}} = \frac{1}{2h}$).

In the following discussion, we prove that the function $F^\alpha(x)$ interpolates the given data.

Step 1. If $\text{mod}(k, M) = 0$, i.e., if k is a multiple of M , then

$$\sum_{n=-M/2+1}^{M/2} \exp\left(2\pi i \frac{nk}{M}\right) = M. \quad (2.20)$$

If, instead, $\text{mod}(k, M) \neq 0$, then

$$\left[1 - \exp\left(2\pi i \frac{k}{M}\right)\right] \sum_{n=-M/2+1}^{M/2} \exp\left(2\pi i \frac{nk}{M}\right) = \quad (2.21)$$

$$= \sum_{n=-M/2+1}^{M/2} \exp\left(2\pi i \frac{nk}{M}\right) - \sum_{n=-M/2+1}^{M/2} \exp\left(2\pi i \frac{(n+1)k}{M}\right) \quad (2.22)$$

$$= \sum_{n=-M/2+1}^{M/2} \exp\left(2\pi i \frac{nk}{M}\right) \quad (2.23)$$

$$- \sum_{n=-M/2+2}^{M/2} \exp\left(2\pi i \frac{nk}{M}\right) - \exp\left(2\pi i \frac{(M/2+1)k}{M}\right) \quad (2.24)$$

$$= \exp\left(2\pi i \frac{(-M/2+1)k}{M}\right) [1 - \exp(2\pi i k)] = 0 \quad (2.25)$$

i.e.,

$$\sum_{n=-M/2+1}^{M/2} \exp\left(2\pi i \frac{nk}{M}\right) = 0. \quad (2.26)$$

Step 2. Substitution of $x = x_k = kh$ in $F^\alpha(x)$ yields

$$F^\alpha(x_k) = \frac{1}{M} \left[\sum_{n=-M/2+1}^{M/2-1} H_n^\alpha \exp\left(-2\pi i \frac{nk}{M}\right) + H_{M/2}^\alpha \cos(-\pi k) \right] \quad (2.27)$$

$$= \frac{1}{M} \sum_{n=-M/2+1}^{M/2} H_n^\alpha \exp\left(-2\pi i \frac{nk}{M}\right) \quad (2.28)$$

$$= \frac{1}{M} \sum_{n=-M/2+1}^{M/2} \sum_{l=1}^M y_l^\alpha \exp\left(2\pi i \frac{n(l-k)}{M}\right) \quad (2.29)$$

$$= \frac{1}{M} \sum_{l=1}^M y_l^\alpha \sum_{n=-M/2+1}^{M/2} \exp\left(2\pi i \frac{n(l-k)}{M}\right) \quad (2.30)$$

$$= y_k^\alpha, \quad (2.31)$$

where we have used the result from Step 1 in arriving at the last equality.

The Fourier interpolant is periodic in x with period Mh i.e. $F^\alpha(x + Mh) = F^\alpha(x)$. As a remark, it can be shown that the discrete Fourier transform is also periodic with period M .

$$\begin{aligned} H_{n+M}^\alpha &= \sum_{l=1}^M y_l^\alpha \exp\left(2\pi i \frac{(n+M)l}{M}\right) \\ &= \sum_{l=1}^M y_l^\alpha \exp\left(2\pi i \frac{nl}{M}\right) \\ &= H_n^\alpha \end{aligned} \tag{2.32}$$

Moreover, if $*$ denotes a complex conjugate, then

$$(H_n^\alpha)^* = \sum_{l=1}^M y_l^\alpha \exp\left(-2\pi i \frac{nl}{M}\right) = H_{-l}^\alpha \tag{2.33}$$

It follows that

$$\left(H_{M/2}^\alpha\right)^* = H_{-M/2}^\alpha = H_{M/2}^\alpha \tag{2.34}$$

and

$$(H_0^\alpha)^* = H_0^\alpha, \tag{2.35}$$

i.e., $H_{M/2}^\alpha$ and H_0^α are both real. It follows that the Fourier interpolant is also a real function.

2.4 Bicubic Interpolation

In this section, we consider the derivation of the functional form of a bicubic spline for a two-dimensional surface in $\mathbb{R}^N$, i.e., two independent variables $\mathbf{x} = (x_1, x_2)$. The interpolant $F^\alpha(\mathbf{x})$ is a combination of piecewise defined cubic polynomials that interpolate the given data. Specifically, the spline is constructed to ensure $\mathcal{C}^2$ continuity at each interior data point. Given sampled data $(\mathbf{x}_{i,j}, \mathbf{y}_{i,j})$, $i = 0, \dots, M_1$ and $j = 0, \dots, M_2$, such that

$$y_{0,j}^\alpha = y_{M_1,j}^\alpha, \text{ for all } j \tag{2.36}$$

and

$$y_{i,0}^\alpha = y_{i,M_2}^\alpha, \text{ for all } i \tag{2.37}$$

the interpolating function for the surface is given by

$$\begin{aligned} F^\alpha(x_1, x_2) &= F^\alpha(x_{k-1}^1, x_2) p_1(t(x_1)) + F^\alpha(x_k^1, x_2) p_2(t(x_1)) \\ &\quad + h_k^1 \left[\frac{\partial F^\alpha}{\partial x_1}(x_{k-1}^1, x_2) q_1(t(x_1)) + \frac{\partial F^\alpha}{\partial x_1}(x_k^1, x_2) q_2(t(x_1)) \right], \end{aligned} \tag{2.38}$$

for $x_{k-1}^1 \leq x_1 \leq x_k^1$, where

$$\begin{aligned} h_k^1 &= x_k^1 - x_{k-1}^1 \\ t(x_1) &= \frac{x_1 - x_{k-1}^1}{h_k^1} \end{aligned}$$

and p_1, p_2, q_1 , and q_2 as defined previously. Moreover,

$$\begin{aligned} F^\alpha(x_i^1, x_2) &= y_{i,j-1}^\alpha p_1(s(x_2)) + y_{i,j}^\alpha p_2(s(x_2)) \\ &\quad + h_j^2 \left[\frac{\partial F^\alpha}{\partial x_2}(x_i^1, x_{j-1}^2) q_1(t(x_1)) + \frac{\partial F^\alpha}{\partial x_2}(x_i^1, x_j^2) q_2(t(x_1)) \right] \end{aligned} \quad (2.39)$$

$$\begin{aligned} \frac{\partial F^\alpha}{\partial x_1}(x_i^1, x_2) &= \frac{\partial F^\alpha}{\partial x_1}(x_i^1, x_{j-1}^2) p_1(s(x_2)) + \frac{\partial F^\alpha}{\partial x_1}(x_i^1, x_j^2) p_2(s(x_2)) \\ &\quad + h_j^2 \frac{\partial^2 F^\alpha}{\partial x_1 \partial x_2}(x_i^1, x_{j-1}^2) q_1(s(x_2)) \\ &\quad + h_j^2 \frac{\partial^2 F^\alpha}{\partial x_1 \partial x_2}(x_i^1, x_j^2) q_2(s(x_2)) \end{aligned} \quad (2.40)$$

for $x_{j-1}^2 \leq x_2 \leq x_j^2$, where

$$h_j^2 = x_j^2 - x_{j-1}^2 \quad (2.41)$$

$$s(x_2) = \frac{x_2 - x_{j-1}^2}{h_j^2} \quad (2.42)$$

As in a previous section, the imposition of $\mathcal{C}^2$ continuity at all internal nodes and across the end nodes (recalling the periodicity) yields linear equations in the unknown coefficients

$$\frac{\partial F^\alpha}{\partial x_1}(x_i^1, x_j^2), \frac{\partial F^\alpha}{\partial x_2}(x_i^1, x_j^2), \text{ and } \frac{\partial^2 F^\alpha}{\partial x_1 \partial x_2}(x_i^1, x_j^2) \quad (2.43)$$

in terms of $\mathbf{x}_{i,j}$ and $\mathbf{y}_{i,j}$ for $i = 1, \dots, M_1$ and $j = 1, \dots, M_2$.

2.5 Geometric Properties of Curves and Surfaces

The interpolating functions obtained above provide parametric representations of the corresponding geometric object. In this section we briefly look into some of the properties of curves and two-dimensional surfaces in a three-dimensional space and indicate how the interpolant can be used to quantify these. A more elaborate discussion of the properties can be found in reference [11].

2.5.1 Properties of a Curve

Consider a curve $\mathcal{C}$ in a three-dimensional space and let $\mathbf{y}(x_1)$ be an *allowable parametric representation*, such that

$$\sigma(x_1) = \sqrt{\frac{d\mathbf{y}}{dx_1}(x_1) \bullet \frac{d\mathbf{y}}{dx_1}(x_1)} > 0 \quad (2.44)$$

Then, it is possible to reparametrize the curve in terms of the arc length s , such that

$$\frac{ds}{dx_1}(x_1) = \sigma(x_1) \quad (2.45)$$

The parametric representation $\tilde{\mathbf{y}}(s) = \mathbf{y}(x_1(s))$ is called the curve's *natural parametric representation*.

The *tangent vector*

$$\mathbf{t}(s) = \frac{d\tilde{\mathbf{y}}}{ds}(s) = \frac{d\mathbf{y}}{dx_1}(x_1(s)) / \frac{ds}{dx_1}(x_1(s)) = \frac{1}{\sigma(x_1(s))} \frac{d\mathbf{y}}{dx_1}(x_1(s)) \quad (2.46)$$

is then a unit vector tangential to the curve at the point $\tilde{\mathbf{y}}(s)$. The set of all vectors that are perpendicular to the tangent vector at the point $\tilde{\mathbf{y}}(s)$ span a plane through $\tilde{\mathbf{y}}(s)$ called the *normal plane* to the curve at $\tilde{\mathbf{y}}(s)$.

Similarly, if $\mathbf{t}(s) \bullet \mathbf{t}(s) > 0$, then the *curvature vector*

$$\mathbf{p}(s) = \frac{1}{\kappa(s)} \frac{d\mathbf{t}}{ds}(s) \quad (2.47)$$

where

$$\kappa(s) = \sqrt{\mathbf{t}(s) \bullet \mathbf{t}(s)} \quad (2.48)$$

is a unit vector perpendicular to the curve at the point $\tilde{\mathbf{y}}(s)$, since

$$\mathbf{t}(s) \bullet \mathbf{t}(s) = 1 \Rightarrow \frac{d\mathbf{t}}{ds}(s) \bullet \mathbf{t}(s) = 0 \quad (2.49)$$

The tangent vector and the curvature vector span a plan through $\tilde{\mathbf{y}}(s)$ called the *osculating plane*.

The quantity $\kappa(s)$ is known as the *curvature* of the curve at the point $\tilde{\mathbf{y}}(s)$. Its reciprocal

$$\rho(s) = \frac{1}{\kappa(s)} \quad (2.50)$$

is called the *radius of curvature* of the curve at the point $\tilde{\mathbf{y}}(s)$ and corresponds to the radius of the circle in the osculating plane that achieves second-order tangential contact with the curve at the point $\tilde{\mathbf{y}}(s)$.

Given a parametric representation of a curve as provided by a cubic spline interpolant or by a Fourier interpolant, the above methodology may thus be employed to estimate the local curvature given the sampled data.

2.5.2 Properties of a Surface

Consider a two-dimensional surface $\mathcal{S}$ in a three-dimensional space and let $\mathbf{y}(x_1, x_2)$ be an *allowable parametric representation*, such that

$$\mathbf{y}_{,1}(x_1, x_2) = \frac{\partial \mathbf{y}}{\partial x_1}(x_1, x_2) \text{ and } \mathbf{y}_{,2}(x_1, x_2) = \frac{\partial \mathbf{y}}{\partial x_2}(x_1, x_2) \quad (2.51)$$

are linearly independent for all x_1 and x_2 . Then, the tangent vector to the curve

$$\tilde{\mathbf{y}}(s) = \mathbf{y}(x_1(s), x_2(s)) \quad (2.52)$$

at the point $\tilde{\mathbf{y}}(s)$ is given by

$$\mathbf{t}(s) = \frac{d\tilde{\mathbf{y}}}{ds}(s) = \sum_{i=1}^2 \frac{\partial \mathbf{y}}{\partial x^i}(x_1(s), x_2(s)) \frac{dx_i}{ds}(s) \quad (2.53)$$

such that

$$\mathbf{t}(s) \bullet \mathbf{t}(s) = \sum_{i,j=1}^2 g_{ij}(x_1(s), x_2(s)) \frac{dx_i}{ds}(s) \frac{dx_j}{ds}(s) = 1 \quad (2.54)$$

where

$$g_{ij}(x_1(s), x_2(s)) = \mathbf{y}_{,i}(x_1(s), x_2(s)) \bullet \mathbf{y}_{,j}(x_1(s), x_2(s)) \quad (2.55)$$

It follows that the vectors $\mathbf{y}_{,1}(x_1, x_2)$ and $\mathbf{y}_{,2}(x_1, x_2)$ span the tangent plane to the surface at the point $\mathbf{y}(x_1, x_2)$. The vector

$$\mathbf{n}(x_1, x_2) = \frac{\mathbf{y}_{,1}(x_1, x_2) \times \mathbf{y}_{,2}(x_1, x_2)}{\|\mathbf{y}_{,1}(x_1, x_2) \times \mathbf{y}_{,2}(x_1, x_2)\|} \quad (2.56)$$

is called the *unit normal vector* to the surface at the point $\mathbf{y}(x_1, x_2)$. The vectors $\mathbf{y}_{,1}(x_1, x_2)$, $\mathbf{y}_{,2}(x_1, x_2)$, and $\mathbf{n}(x_1, x_2)$ form a basis for space.

Now denote by $\gamma(s)$ the angle between $\mathbf{n}(x_1(s), x_2(s))$ and the curvature vector $\mathbf{p}(s)$ at the point $\tilde{\mathbf{y}}(s) = \mathbf{y}(x_1(s), x_2(s))$. Then, since

$$\mathbf{p}(s) = \frac{1}{\kappa(s)} \frac{d^2 \tilde{\mathbf{y}}}{ds^2} = \sum_{i,j=1}^2 \mathbf{y}_{,ij}(x_1(s), x_2(s)) \frac{dx_i}{ds}(s) \frac{dx_j}{ds}(s) + \sum_{i=1}^2 \mathbf{y}_{,i}(x_1(s), x_2(s)) \frac{d^2 x_i}{ds^2}(s), \quad (2.57)$$

where

$$\mathbf{y}_{,ij}(x_1(s), x_2(s)) = \frac{\partial^2 \mathbf{y}}{\partial x^i \partial x^j}(x_1(s), x_2(s)) \quad (2.58)$$

it follows that

$$\kappa_n(s) \stackrel{def}{=} \kappa(s) \cos \gamma(s) = \sum_{i,j=1}^2 b_{ij}(x_1(s), x_2(s)) \frac{dx_i}{ds}(s) \frac{dx_j}{ds}(s) \quad (2.59)$$

where

$$b_{ij}(x_1(s), x_2(s)) = \mathbf{y}_{,ij}(x_1(s), x_2(s)) \bullet \mathbf{n}(x_1(s), x_2(s)) \quad (2.60)$$

Combining equations (2.54) and (2.59) then yields

$$\sum_{i,j=1}^2 (g_{ij}(x_1(s), x_2(s)) \kappa_n(s) - b_{ij}(x_1(s), x_2(s))) \frac{dx_i}{ds}(s) \frac{dx_j}{ds}(s) = 0 \quad (2.61)$$

Different curves through the point $\tilde{\mathbf{y}}(s) = \mathbf{y}(x_1(s), x_2(s))$ correspond to different parametrizations $x_1(s)$ and $x_2(s)$, to different derivatives $\frac{dx_1}{ds}(s)$ and $\frac{dx_2}{ds}(s)$, and to different curvatures $\kappa_n(s)$. To find the extremal values for κ_n at the point $\mathbf{y}(x_1, x_2)$, differentiate (2.61) with respect to $\frac{dx_1}{ds}$ and $\frac{dx_2}{ds}$ to obtain

$$\begin{aligned} 2(g_{11}(x_1, x_2) \kappa_n - b_{11}(x_1, x_2)) \frac{dx_1}{ds}(s) + 2(g_{12}(x_1, x_2) \kappa_n - b_{12}(x_1, x_2)) \frac{dx_2}{ds}(s) &= 0 \\ 2(g_{12}(x_1, x_2) \kappa_n - b_{12}(x_1, x_2)) \frac{dx_1}{ds}(s) + 2(g_{22}(x_1, x_2) \kappa_n - b_{22}(x_1, x_2)) \frac{dx_2}{ds}(s) &= 0 \end{aligned} \quad (2.62)$$

A nontrivial solution to these equations exists when

$$\begin{vmatrix} g_{11}(x_1, x_2) \kappa_n - b_{11}(x_1, x_2) & g_{12}(x_1, x_2) \kappa_n - b_{12}(x_1, x_2) \\ g_{12}(x_1, x_2) \kappa_n - b_{12}(x_1, x_2) & g_{22}(x_1, x_2) \kappa_n - b_{22}(x_1, x_2) \end{vmatrix} = 0 \quad (2.63)$$

the roots of which yield the extremal values of κ_n . These are known as the *principal curvatures* of the surface at the point $\mathbf{y}(x_1, x_2)$. The corresponding tangent directions obtained by substitution of $\frac{dx_1}{ds}(s)$ and $\frac{dx_2}{ds}(s)$ for the extremal values of κ_n then yield the *principal directions of curvature* of the surface at the point $\mathbf{y}(x_1, x_2)$. As an example, a point is said to be *elliptic* if both principal curvatures are positive, to be *hyperbolic* if the principal curvatures have opposite sign, and *parabolic* if one of the principal curvatures is zero. For an elliptic point, the principal curvatures correspond to the inverses of the radii of the ellipsoid that achieves second-order tangential contact with the surface at the point $\mathbf{y}(x_1, x_2)$.

Given a parametric representation of a surface as provided by a bicubic spline interpolant, the above methodology may thus be employed to estimate the principal curvatures and the principal directions of curvature given the sampled data.

Chapter 3

Continuation of Invariant Tori

In this chapter, we propose an iterative method for the continuation of invariant tori of maps. Specifically, we illustrate the method for the continuation of invariant one-dimensional and two-dimensional tori. The algorithm is then generalized to locate an n -dimensional invariant torus of a m -dimensional map. For successful convergence, the algorithm requires a good initial guess and uses a Newton-Raphson-based scheme to converge on the invariant torus. For a map parameterized by a vector of system parameters μ , convergence of the iterative scheme for some value μ_0 yields an initial guess for the continuation of the invariant torus under variations in μ , although some preprocessing may be necessary. The method proposed is not affected by the stability of the torus under iterations of the map and hence can be used to trace branches of both stable and unstable tori.

The proposed method is general in nature and one can use it even for continuous dynamical systems, if we define the map in terms of the corresponding flow function. For example, we integrate the system of equations until a certain event has occurred. This event could correspond to the intersection of a system orbit with a Poincaré section (Poincaré map ref: 1.2.2) or to integration for a fixed time interval.

3.1 One-Dimensional Tori

Consider a discrete dynamical system given by an m -dimensional map $\mathbf{f}$ and assume that there exists an invariant closed curve $\mathcal{C}$ in state space, i.e., such that $\mathbf{f}(\mathcal{C}) = \mathcal{C}$. Further assume that $\mathcal{C}$ is at least C^1 continuous. Specifically, there exists a periodic C^1 parametrization $\tilde{\mathbf{x}}(t)$, such that $\tilde{\mathbf{x}}(t + 2\pi p) = \tilde{\mathbf{x}}(t)$ for an arbitrary integer p . The goal of this section is to formulate an algorithm for locating this invariant curve for a given choice of system parameters and for continuing the invariant curve under small variations in the system parameters.

Consider an arbitrary point $\tilde{\mathbf{x}}(t)$ on the invariant curve. This point is mapped by $\mathbf{f}$ to the point $\mathbf{f}(\tilde{\mathbf{x}}(t))$, which also lies on the invariant curve (cf. Fig. 3.1), i.e., $\mathbf{f}(\tilde{\mathbf{x}}(t)) = \tilde{\mathbf{x}}(t + \tilde{\tau}(t))$. Specifically, the *shift* $\tilde{\tau}(t)$ is a periodic function of t with period 2π that corresponds to the difference in the value of the parameter t for the points $\tilde{\mathbf{x}}(t)$ and $\mathbf{f}(\tilde{\mathbf{x}}(t))$. In what follows, $\tilde{\tau}(t)$ is assumed to be C^1 .

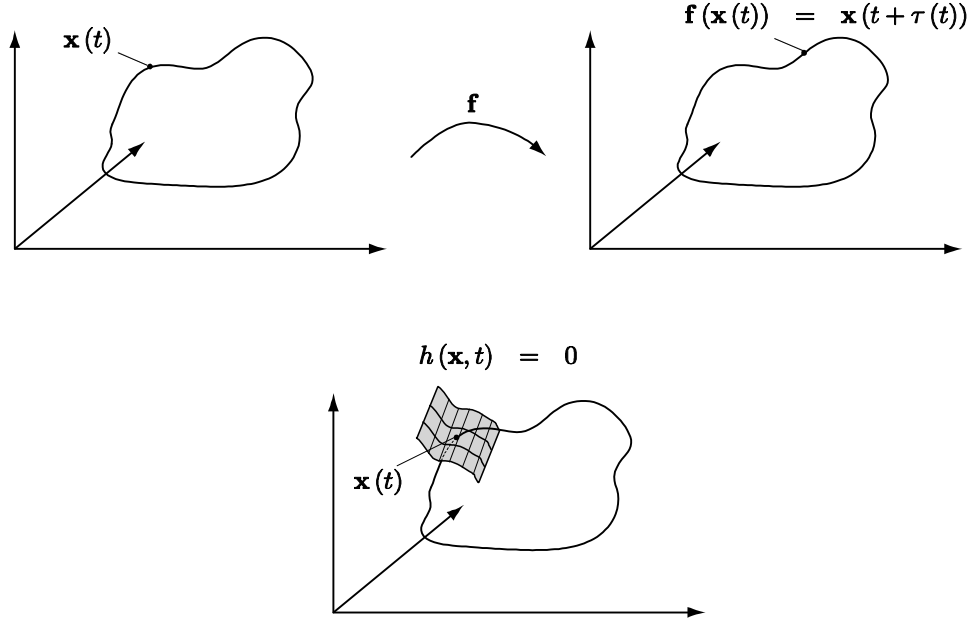


Figure 3.1: The invariant 1-D Tori in 3-D state space

It is possible to introduce a family of event functions $h(\mathbf{x}, t)$ parameterized by t , such that, for each t , the corresponding zero-level surface contains the point $\tilde{\mathbf{x}}(t)$ and is transversal to the invariant curve at the point $\tilde{\mathbf{x}}(t)$. Since state space is assumed to be m -dimensional, these zero-level surfaces are $m - 1$ -dimensional, i.e., local co-dimension-one hypersurfaces that intersect the invariant curve transversally at the corresponding point along the curve.

From the above construction, it follows that the parametrization $\tilde{\mathbf{x}}(t)$ of the invariant curve and the shift $\tilde{\tau}(t)$ satisfy the following set of *infinite-dimensional* nonlinear equations:

$$\begin{aligned} \mathbf{f}(\mathbf{x}(t)) - \mathbf{x}(t + \tau(t)) &= \mathbf{0} \\ h(\mathbf{x}(t), t) &= 0 \end{aligned} \tag{3.1}$$

To discretize these equations, consider the finite subset of equations obtained by restricting attention to a discrete set of values of $t \in \{t_k\}_{k=1}^N$, where $t_N = 2\pi$:

$$\begin{aligned} \mathbf{f}(\mathbf{x}_k) - \mathbf{x}(t_k + \tau_k) &= \mathbf{0} \\ h_k(\mathbf{x}_k) &= 0 \end{aligned}$$

where

$$\mathbf{x}_k = \mathbf{x}(t_k), \tau_k = \tau(t_k) \quad \text{and} \quad h_k(\mathbf{x}) = h(\mathbf{x}, t_k)$$

To complete the reduction to a finite-dimensional problem, we replace the function $\mathbf{x}(t)$ in the first equation with a suitably chosen interpolant $\mathbf{X}(t; \mathbf{x}_1, \dots, \mathbf{x}_N)$ through the points $\mathbf{x}_1, \dots, \mathbf{x}_N$:

$$\mathbf{f}(\mathbf{x}_k) - \mathbf{X}(t_k + \tau_k; \mathbf{x}_1, \dots, \mathbf{x}_N) = \mathbf{0} \quad (3.2a)$$

$$h_k(\mathbf{x}_k) = 0 \quad (3.2b)$$

The discrete formulation results in $mN + N$ equations in the $mN + N$ unknowns $\mathbf{x}_k$ and τ_k for $k = 1, \dots, N$. Thus, the system of equations is well-posed and one can attempt to solve for the unknowns. Specifically, if we denote by $\mathbf{y}$ the collection of unknowns and by $\mathbf{F}(\mathbf{y})$ the collection of left-hand sides of the discretized equations, the iterative map

$$\mathbf{y}^{(n+1)} = \mathbf{y}^{(n)} - \left[\mathbf{F}_{\mathbf{y}}(\mathbf{y}^{(n)}) \right]^{-1} \mathbf{F}(\mathbf{y}^{(n)})$$

corresponds to the Newton-Raphson method for finding a root $\mathbf{y}^*$ to the equation $\mathbf{F}(\mathbf{y}) = \mathbf{0}$ given a good initial guess $\mathbf{y}^{(0)} \approx \mathbf{y}^*$. Here, $\mathbf{F}_{\mathbf{y}}$ denotes the Jacobian matrix of $\mathbf{F}$ with respect to its vector of arguments.

A root $\mathbf{y}^*$ to the equation $\mathbf{F}(\mathbf{y}) = \mathbf{0}$ implies the existence of a curve $\mathbf{X}^*(t; \mathbf{x}_1^*, \dots, \mathbf{x}_N^*)$ that interpolates a set of N data points in space $\mathbf{x}_1^*, \dots, \mathbf{x}_N^*$, such that

$$\mathbf{X}^*(t_k; \mathbf{x}_1^*, \dots, \mathbf{x}_N^*) = \mathbf{x}_k^*$$

and such that

$$\mathbf{f}(\mathbf{x}_k^*) = \mathbf{X}^*(t_k + \tau_k^*; \mathbf{x}_1^*, \dots, \mathbf{x}_N^*),$$

i.e., the data points are mapped to some other point on the interpolant. This does not imply, however, that *every* point on the interpolant is mapped to some other point on the interpolant. Thus, the interpolant **need not** satisfy the original infinite-dimensional problem. Similarly, if $\tilde{\mathbf{x}}(t)$ and $\tilde{\tau}(t)$ is a solution to the original infinite-dimensional problem, such that $\tilde{\mathbf{x}}_k = \tilde{\mathbf{x}}(t_k)$ and $\tilde{\tau}_k = \tilde{\tau}(t_k)$, then it is **not** necessary the case that

$$\mathbf{f}(\tilde{\mathbf{x}}_k) = \mathbf{X}(t_k + \tilde{\tau}_k; \tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_N),$$

where $\mathbf{X}$ is some interpolant through the points $\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_N$. After all, the choice of interpolant must be made independently of the unknown invariant curve and will typically deviate from the original curve.

To improve the agreement between the solution to the discretized problem and that of the original infinite-dimensional problem, one is lead to consider increasingly more finely resolved discretizations. In doing so, it is important to recognize the dependence of the

result on the choice of interpolant and the parametrization of the interpolant. For example, when using cubic splines, different choices of grid values t_k for the same spatial data points will result in different interpolants. Retaining unchanging values of the grid values under parameter variations while the spatial data points cluster may introduce artificial oscillations of the interpolant and a resultant lack of convergence. We return to these issues in the following chapter.

A special case of the above algorithm is obtained when the invariant curve can be periodically parametrized by one of the state variables and when the shift is known *a priori*. Specifically, suppose that the state $\mathbf{x}$ can be decomposed as $\mathbf{x} = \begin{pmatrix} \theta & \mathbf{r} \end{pmatrix}^T$, such that the map $\mathbf{f}$ reduces to the form

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_\theta(\mathbf{r}, \theta) \bmod 2\pi \\ \mathbf{f}_r(\mathbf{r}, \theta) \end{pmatrix}$$

In this case, the shift is given by

$$\tilde{\tau}(\theta) = f_\theta(\tilde{\mathbf{r}}(\theta), \theta) - \theta \bmod 2\pi$$

The infinite-dimensional formulation is now given by

$$\mathbf{f}_r(\mathbf{r}(\theta), \theta) - \mathbf{r}(f_\theta(\mathbf{r}(\theta), \theta)) = \mathbf{0}$$

Furthermore, the discretized version becomes

$$\mathbf{f}_r(\mathbf{r}_k, \theta_k) - \mathbf{R}(f_\theta(\mathbf{r}_k, \theta_k); \mathbf{r}_1, \dots, \mathbf{r}_N) = \mathbf{0} \quad (3.3)$$

where $\mathbf{R}$ is an interpolant, such that

$$\mathbf{R}(\theta_k; \mathbf{r}_1, \dots, \mathbf{r}_N) = \mathbf{r}_k$$

This yields $(m-1)N$ equations in $(m-1)N$ unknowns and can again be approached using the Newton method.

3.2 Two-Dimensional Tori

Now consider a discrete dynamical system given by an m -dimensional map $\mathbf{f}$ and assume that there exists an invariant two-dimensional torus $\mathcal{T}$ in state space, i.e., such that $\mathbf{f}(\mathcal{T}) = \mathcal{T}$. Further assume that $\mathcal{T}$ is at least C^1 continuous. Specifically, there exists a periodic C^1 parametrization $\tilde{\mathbf{x}}(t, s)$, such that $\tilde{\mathbf{x}}(t + 2\pi p, s + 2\pi q) = \tilde{\mathbf{x}}(t, s)$ for arbitrary integers p and q . The goal of this section is to formulate an algorithm for locating this invariant torus for a given choice of system parameters and for continuing the invariant torus under small variations in the system parameters.

Consider an arbitrary point $\tilde{\mathbf{x}}(t, s)$ on the invariant torus. This point is mapped by $\mathbf{f}$ to the point $\mathbf{f}(\tilde{\mathbf{x}}(t, s))$, which also lies on the invariant torus (cf. Fig. 3.2), i.e., $\mathbf{f}(\tilde{\mathbf{x}}(t, s)) = \tilde{\mathbf{x}}(t + \tilde{\tau}(t, s), s + \tilde{\sigma}(t, s))$. Specifically, the *shifts* $\tilde{\tau}(t, s)$ and $\tilde{\sigma}(t, s)$ are periodic functions of t and s with periods 2π that correspond to the difference in the value of the parameter t and s for the points $\tilde{\mathbf{x}}(t, s)$ and $\mathbf{f}(\tilde{\mathbf{x}}(t, s))$. In what follows, $\tilde{\tau}(t, s)$ and $\tilde{\sigma}(t, s)$ are assumed to be C^1 .

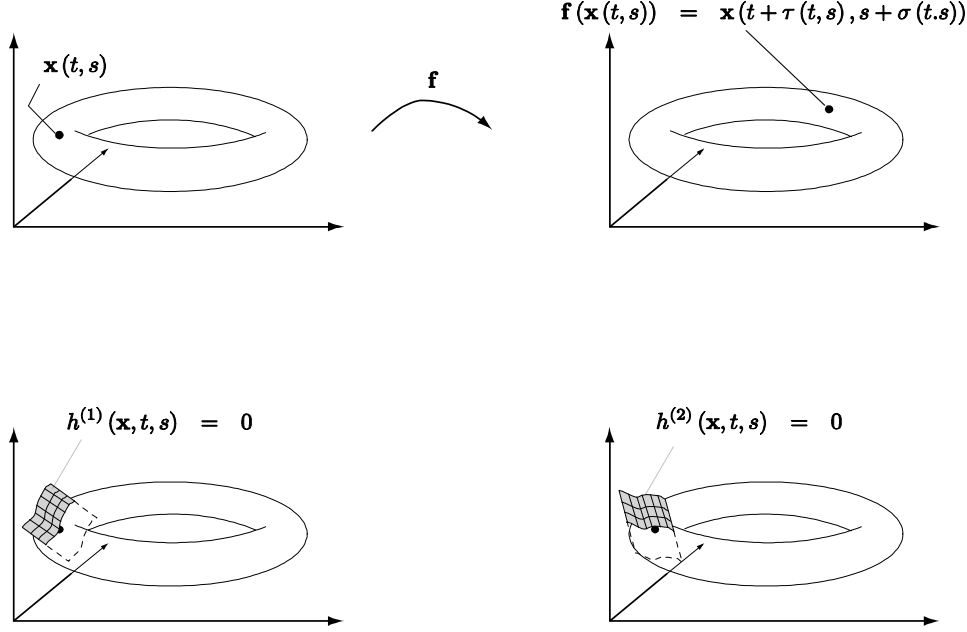


Figure 3.2: The invariant 2-D Tori in 3-D state space

It is possible to introduce two families of functionally independent event functions $h^{(1)}(\mathbf{x}, t, s)$ and $h^{(2)}(\mathbf{x}, t, s)$ parameterized by t and s , such that, for each t and s , the corresponding zero-level surfaces contain the point $\tilde{\mathbf{x}}(t, s)$ and are transversal to the invariant torus at the point $\tilde{\mathbf{x}}(t, s)$. Since state space is assumed to be m -dimensional, these zero-level surfaces are $m - 1$ -dimensional, i.e., local co-dimension-one hypersurfaces that intersect the invariant torus transversally at the corresponding point on the torus. From independence, it follows that their intersection is a local co-dimension-two hypersurface that intersects the invariant torus transversally at the corresponding point on the torus.

From the above construction, it follows that the parametrization $\tilde{\mathbf{x}}(t, s)$ of the invariant torus and the shifts $\tilde{\tau}(t, s)$ and $\tilde{\sigma}(t, s)$ satisfy the following set of *infinite-dimensional* non-

linear equations:

$$\mathbf{f}(\mathbf{x}(t, s)) - \mathbf{x}(t + \tau(t, s), s + \sigma(t, s)) = \mathbf{0} \quad (3.4)$$

$$h^{(1)}(\mathbf{x}(t, s), t, s) = 0 \quad (3.5)$$

$$h^{(2)}(\mathbf{x}(t, s), t, s) = 0 \quad (3.6)$$

To discretize these equations, consider the finite subset of equations obtained by restricting attention to a discrete set of values of $t \in \{t_k\}_{k=1}^N$ and $s \in \{s_k\}_{k=1}^M$, where $t_N = 2\pi$ and $s_M = 2\pi$:

$$\mathbf{f}(\mathbf{x}_{k_1, k_2}) - \mathbf{x}(t_{k_1} + \tau_{k_1, k_2}, s_{k_2} + \sigma_{k_1, k_2}) = \mathbf{0}$$

$$h_{k_1, k_2}^{(1)}(\mathbf{x}_{k_1, k_2}) = 0$$

$$h_{k_1, k_2}^{(2)}(\mathbf{x}_{k_1, k_2}) = 0$$

where

$$\begin{aligned} \mathbf{x}_{k_1, k_2} &= \mathbf{x}(t_{k_1}, s_{k_2}), \tau_{k_1, k_2} = \tau(t_{k_1}, s_{k_2}), \sigma_{k_1, k_2} = \sigma(t_{k_1}, s_{k_2}), \\ h_{k_1, k_2}^{(1)}(\mathbf{x}) &= h^{(1)}(\mathbf{x}, t_{k_1}, s_{k_2}) \text{ and } h_{k_1, k_2}^{(2)}(\mathbf{x}) = h^{(2)}(\mathbf{x}, t_{k_1}, s_{k_2}) \end{aligned}$$

To complete the reduction to a finite-dimensional problem, we replace the function $\mathbf{x}(t, s)$ in the first equation with a suitably chosen interpolant $\mathbf{X}(t, s; \mathbf{x}_{1,1}, \dots, \mathbf{x}_{N,M})$ through the points $\mathbf{x}_{1,1}, \dots, \mathbf{x}_{N,M}$:

$$\mathbf{f}(\mathbf{x}_{k_1, k_2}) - \mathbf{X}(t_{k_1} + \tau_{k_1, k_2}, s_{k_2} + \sigma_{k_1, k_2}; \mathbf{x}_{1,1}, \dots, \mathbf{x}_{N,M}) = \mathbf{0} \quad (3.7a)$$

$$h_{k_1, k_2}^{(1)}(\mathbf{x}_{k_1, k_2}) = 0 \quad (3.7b)$$

$$h_{k_1, k_2}^{(2)}(\mathbf{x}_{k_1, k_2}) = 0 \quad (3.7c)$$

The discrete formulation results in $mNM + 2NM$ equations in the $mNM + 2NM$ unknowns $\mathbf{x}_{k_1, k_2}$, τ_{k_1, k_2} , and σ_{k_1, k_2} for $k_1 = 1, \dots, N$ and $k_2 = 1, \dots, M$. Thus, the system of equations is well-posed and one can again attempt to solve for the unknowns. Specifically, if we denote by $\mathbf{y}$ the collection of unknowns and by $\mathbf{F}(\mathbf{y})$ the collection of left-hand sides of the discretized equations, the iterative map

$$\mathbf{y}^{(n+1)} = \mathbf{y}^{(n)} - \left[\mathbf{F}_{\mathbf{y}}(\mathbf{y}^{(n)}) \right]^{-1} \mathbf{F}(\mathbf{y}^{(n)})$$

corresponds to the Newton-Raphson method for finding a root $\mathbf{y}^*$ to the equation $\mathbf{F}(\mathbf{y}) = \mathbf{0}$ given a good initial guess $\mathbf{y}^{(0)} \approx \mathbf{y}^*$.

A special case of the above algorithm is obtained when the invariant torus can be periodically parametrized by one or two of the state variables and when the corresponding shifts are

known *a priori*. Specifically, suppose that the state $\mathbf{x}$ can be decomposed as $\mathbf{x} = \begin{pmatrix} \theta & \mathbf{r} \end{pmatrix}^T$, such that the map $\mathbf{f}$ reduces to the form

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_\theta(\mathbf{r}, \theta) \bmod 2\pi \\ \mathbf{f}_r(\mathbf{r}, \theta) \end{pmatrix}$$

In this case, the shift is given by

$$\hat{\tau}(\theta, s) = f_\theta(\tilde{\mathbf{r}}(\theta, s), \theta) - \theta \bmod 2\pi$$

The infinite-dimensional formulation is now given by

$$\begin{aligned} \mathbf{f}_r(\mathbf{r}(\theta, s), \theta) - \mathbf{r}(f_\theta(\mathbf{r}(\theta, s), \theta), s + \sigma(\theta, s)) &= \mathbf{0} \\ h(\mathbf{r}(\theta, s), \theta, s) &= 0 \end{aligned}$$

Furthermore, the discretized version becomes

$$\begin{aligned} \mathbf{f}_r(\mathbf{r}_{k_1, k_2}, \theta_{k_1}) - \mathbf{R}(f_\theta(\mathbf{r}_{k_1, k_2}, \theta_{k_1}), s_{k_2} + \sigma_{k_1, k_2}; \mathbf{r}_{1,1}, \dots, \mathbf{r}_{N,M}) &= \mathbf{0} \\ h_{k_1, k_2}(\mathbf{r}_{k_1, k_2}) &= 0 \end{aligned}$$

where $\mathbf{R}$ is an interpolant, such that

$$\mathbf{R}(\theta_{k_1}, s_{k_2}; \mathbf{r}_{1,1}, \dots, \mathbf{r}_{N,M}) = \mathbf{r}_{k_1, k_2}$$

This yields $(m-1)NM + NM$ equations in $(m-1)NM + NM$ unknowns and can again be approached using the Newton method.

Similarly, if the state $\mathbf{x}$ can be decomposed as $\mathbf{x} = \begin{pmatrix} \theta & \varphi & \mathbf{r} \end{pmatrix}^T$, such that the map $\mathbf{f}$ reduces to the form

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_\theta(\mathbf{r}, \theta, \varphi) \bmod 2\pi \\ f_\varphi(\mathbf{r}, \theta, \varphi) \bmod 2\pi \\ \mathbf{f}_r(\mathbf{r}, \theta, \varphi) \end{pmatrix}$$

In this case, the shifts are given by

$$\begin{aligned} \tilde{\tau}(\theta, \varphi) &= f_\theta(\tilde{\mathbf{r}}(\theta, \varphi), \theta, \varphi) - \theta \bmod 2\pi \\ \tilde{\sigma}(\theta, \varphi) &= f_\varphi(\tilde{\mathbf{r}}(\theta, \varphi), \theta, \varphi) - \varphi \bmod 2\pi \end{aligned}$$

The infinite-dimensional formulation is now given by

$$\mathbf{f}_r(\mathbf{r}(\theta, \varphi), \theta, \varphi) - \mathbf{r}(f_\theta(\mathbf{r}(\theta, \varphi), \theta, \varphi), f_\varphi(\mathbf{r}(\theta, \varphi), \theta, \varphi)) = \mathbf{0}.$$

Furthermore, the discretized version becomes

$$\mathbf{f}_r(\mathbf{r}_{k_1, k_2}, \theta_{k_1}, \varphi_{k_2}) - \mathbf{R}(f_\theta(\mathbf{r}_{k_1, k_2}, \theta_{k_1}, \varphi_{k_2}), f_\varphi(\mathbf{r}_{k_1, k_2}, \theta_{k_1}, \varphi_{k_2}); \mathbf{r}_{1,1}, \dots, \mathbf{r}_{N,M}) = \mathbf{0}$$

where $\mathbf{R}$ is an interpolant, such that

$$\mathbf{R}(\theta_{k_1}, \varphi_{k_2}; \mathbf{r}_{1,1}, \dots, \mathbf{r}_{N,M}) = \mathbf{r}_{k_1, k_2}$$

This yields $(m-2)NM$ equations in $(m-2)NM$ unknowns and can again be approached using the Newton method.

3.3 n -Dimensional Tori

Now consider generalizing the proposed methodology to a discrete dynamical system given by an m -dimensional map $\mathbf{f}$, for which there exists an invariant n -dimensional torus $\mathcal{T}$ in state space, i.e., such that $\mathbf{f}(\mathcal{T}) = \mathcal{T}$. Further assume that $\mathcal{T}$ is at least C^1 continuous. Specifically, there exists a periodic C^1 parametrization $\tilde{\mathbf{x}}(\mathbf{t})$, such that $\tilde{\mathbf{x}}(\mathbf{t} + 2\pi\mathbf{p}) = \tilde{\mathbf{x}}(\mathbf{t})$ for an arbitrary vector of integers $\mathbf{p}$.

Consider an arbitrary point $\tilde{\mathbf{x}}(\mathbf{t})$ on the invariant torus. This point is mapped by $\mathbf{f}$ to the point $\mathbf{f}(\tilde{\mathbf{x}}(\mathbf{t}))$, which also lies on the invariant torus, i.e., $\mathbf{f}(\tilde{\mathbf{x}}(\mathbf{t})) = \tilde{\mathbf{x}}(\mathbf{t} + \tilde{\tau}(\mathbf{t}))$. Specifically, the *shift vector* $\tilde{\tau}(\mathbf{t})$ is a periodic function of $\mathbf{t}$ with period 2π in each component of $\mathbf{t}$ that corresponds to the difference in the value of the parameter $\mathbf{t}$ for the points $\tilde{\mathbf{x}}(\mathbf{t})$ and $\mathbf{f}(\tilde{\mathbf{x}}(\mathbf{t}))$. In what follows, $\tilde{\tau}(\mathbf{t})$ is assumed to be C^1 .

It is again possible to introduce a vector-valued family of functionally independent event functions $\mathbf{h}(\mathbf{x}, \mathbf{t})$ parameterized by $\mathbf{t}$, such that, for each $\mathbf{t}$, the corresponding zero-level surfaces contain the point $\tilde{\mathbf{x}}(\mathbf{t})$ and are transversal to the invariant torus at the point $\tilde{\mathbf{x}}(\mathbf{t})$. Since state space is assumed to be m -dimensional, these zero-level surfaces are $m - 1$ -dimensional, i.e., local co-dimension-one hypersurfaces that intersect the invariant torus transversally at the corresponding point on the torus. From independence, it follows that their intersection is a local co-dimension- n hypersurface that intersects the invariant torus transversally at the corresponding point on the torus.

From the above construction, it follows that the parametrization $\tilde{\mathbf{x}}(\mathbf{t})$ of the invariant torus and the shift $\tilde{\tau}(\mathbf{t})$ satisfies the following set of *infinite-dimensional* nonlinear equations:

$$\mathbf{f}(\mathbf{x}(\mathbf{t})) - \mathbf{x}(\mathbf{t} + \tau(\mathbf{t})) = \mathbf{0} \quad (3.8)$$

$$\mathbf{h}(\mathbf{x}(\mathbf{t}), \mathbf{t}) = \mathbf{0} \quad (3.9)$$

To discretize these equations, consider the finite subset of equations obtained by restricting attention to a discrete set of values of $\mathbf{t} \in \{\mathbf{t}_k\}_{k=1}^N$, where $\mathbf{t}_{N_1, \dots, N_n} = (2\pi \ \dots \ 2\pi)$:

$$\begin{aligned} \mathbf{f}(\mathbf{x}_{k_1, \dots, k_n}) - \mathbf{x}(\mathbf{t}_{k_1, \dots, k_n} + \tau_{k_1, \dots, k_n}) &= \mathbf{0} \\ \mathbf{h}_{k_1, \dots, k_n}(\mathbf{x}_{k_1, \dots, k_n}) &= \mathbf{0} \end{aligned}$$

where

$$\begin{aligned} \mathbf{x}_{k_1, \dots, k_n} &= \mathbf{x}(\mathbf{t}_{k_1, \dots, k_n}), \quad \tau_{k_1, \dots, k_n} = \tau(\mathbf{t}_{k_1, \dots, k_n}), \\ \mathbf{h}_{k_1, \dots, k_n}(\mathbf{x}) &= \mathbf{h}(\mathbf{x}, \mathbf{t}_{k_1, \dots, k_n}). \end{aligned}$$

To complete the reduction to a finite-dimensional problem, we replace the function $\mathbf{x}(\mathbf{t})$ in the first equation with a suitably chosen interpolant $\mathbf{X}(\mathbf{t}; \mathbf{x}_{1, \dots, 1}, \dots, \mathbf{x}_{N_1, \dots, N_n})$ through the points $\mathbf{x}_{1, \dots, 1}, \dots, \mathbf{x}_{N_1, \dots, N_n}$:

$$\mathbf{f}(\mathbf{x}_{k_1, \dots, k_n}) - \mathbf{X}(\mathbf{t}_{k_1, \dots, k_n} + \tau_{k_1, \dots, k_n}; \mathbf{x}_{1, \dots, 1}, \dots, \mathbf{x}_{N_1, \dots, N_n}) = \mathbf{0} \quad (3.10)$$

$$\mathbf{h}_{k_1, \dots, k_n}(\mathbf{x}_{k_1, \dots, k_n}) = \mathbf{0} \quad (3.11)$$

The discrete formulation results in $mN_1 \cdots N_m + nN_1 \cdots N_n$ equations in $mN_1 \cdots N_m + nN_1 \cdots N_n$ unknowns $\mathbf{x}_{k_1, \dots, k_n}$ and $\tau_{k_1, \dots, k_n}$ for $k_i = 1, \dots, N_i$. Thus, the system of equations is well-posed and one can again attempt to solve for the unknowns.

Chapter 4

Results

In this chapter, the method proposed in chapter 3 is used for the evaluation of invariant tori in the state space for different discrete dynamical systems. Section 4.1 illustrates two different examples for a one-dimensional invariant torus (a curve) of a two and a three-dimensional map. Section 4.2 shows results for a two-dimensional invariant torus of a three and a four dimensional map. Section 4.3 shows the results obtained from the sweeping approach, which uses the methodology of one-dimensional tori, to locate the two-dimensional tori of the maps discussed in section 4.2.

4.1 One-Dimensional Torus

Consider the following map [5].

$$\begin{aligned}x_{n+1} &= \left(\lambda r_n(1 - r_n) + \epsilon \frac{x_n}{r_n} \right) \left(\frac{x_n}{r_n} \cos \omega - \frac{y_n}{r_n} \sin \omega \right) \\y_{n+1} &= \left(\lambda r_n(1 - r_n) + \epsilon \frac{x_n}{r_n} \right) \left(\frac{y_n}{r_n} \cos \omega + \frac{x_n}{r_n} \sin \omega \right) \\r_n &= \sqrt{x_n^2 + y_n^2}\end{aligned}\tag{4.1}$$

For $\epsilon = 0$, the coordinates are uncoupled and the r coordinate transforms according to the logistic map. The method proposed is employed to locate the invariant curve for the parameter values of $\lambda = 2.5$, $\omega = 0.8$ and $\epsilon > 0$. Continuation is done upon the invariant curve as ϵ is varied.

The above system can be numerically simulated to obtain the invariant curve for the above parameter taking $\epsilon = 0.0$. This invariant curve is sampled at N different points $\mathbf{x}_k^{(0)}$ for $k = 1, \dots, N$ i.e., we convert the infinite-dimensional problem to a finite-dimensional

problem (cf. 3.1). A periodic cubic spline $\mathbf{X} \left(t; \mathbf{x}_1^{(0)}, \dots, \mathbf{x}_N^{(0)} \right)$ is used to interpolate a curve passing through the sampled points. The point on the interpolant closest to the image $\mathbf{f} \left(\mathbf{x}_k^{(0)} \right)$ is used to compute the shift for each point.

The event function $h_k(\mathbf{x}, t)$ at every sampled point is a line perpendicular to the tangent of the interpolant, i.e.,

$$h_k(\mathbf{x}, t) = \left(\mathbf{x} - \mathbf{x}_k^{(0)} \right) \bullet \frac{d\mathbf{X}}{dt} \left(t_k; \mathbf{x}_1^{(0)}, \dots, \mathbf{x}_N^{(0)} \right) \quad (4.2)$$

Once the initial guess points $\mathbf{x}_k^{(0)}(t)$, the initial shift $\tau_k^{(0)}(t)$ and the event surfaces $h_k(\mathbf{x}, t)$ are chosen, the system of equations 3.2 are formulated and the Newton-Raphson method is used to locate the invariant tori. The converged results are resampled in such a way that the points are uniformly spaced, i.e, equal arc length, on the interpolant passing through the converged points. These resampled point are taken as an initial guess and the shifts are recomputed. The Newton-Raphson formulation is re-run in the hope that we are closer to the actual invariant curve. This also prevents clustering of the sampled points, which may result in a oscillatory interpolant and lack of convergence.

For the results shown in fig. 4.1, the initial guess curve is sampled at 20 equally spaced points. Continuation of the invariant curve is done as the system parameter ϵ is increased. As ϵ is increased, the curve becomes irregular. In order to compute the approximate solution the number of sampled points $\mathbf{x}_k^{(0)}$ is increased. The number of sampled points on the guess curve for $\epsilon = 0.315$ is 110.

Rewriting the above map in polar coordinates yields.

$$\begin{aligned} \theta_{n+1} &= \theta_n + \omega \bmod 2\pi \\ r_{n+1} &= \lambda r_n (1 - r_n) + \epsilon \cos \theta_n \end{aligned} \quad (4.3)$$

For some values of the parameters, it is evident that the radial coordinate can be written as a function of the angular coordinate along the invariant curve. Thus, the number of unknowns for this system can be reduced if we take θ as the parameter. As described in section 3.1, the *shift* in this parameter direction is known. The system of equations reduces to a single functional equation as proposed by Kevrekidis [5]. The results obtained for this system both in cartesian and polar coordinates are shown in figure 4.1.

Now consider a discrete dynamical system given by the map

$$\mathbf{f}(\mathbf{x}) = \phi(\mathbf{x}, 1) \quad (4.4)$$

where $\phi(\mathbf{x}, 1)$ is the *flow* function of the following continuous dynamical system

$$\begin{aligned} \dot{x}_1 &= mx_1 - x_2 - x_1 x_3 \\ \dot{x}_2 &= mx_2 + x_1 \\ \dot{x}_3 &= -x_3 + x_2^2 + x_1^2 x_3 \end{aligned} \quad (4.5)$$

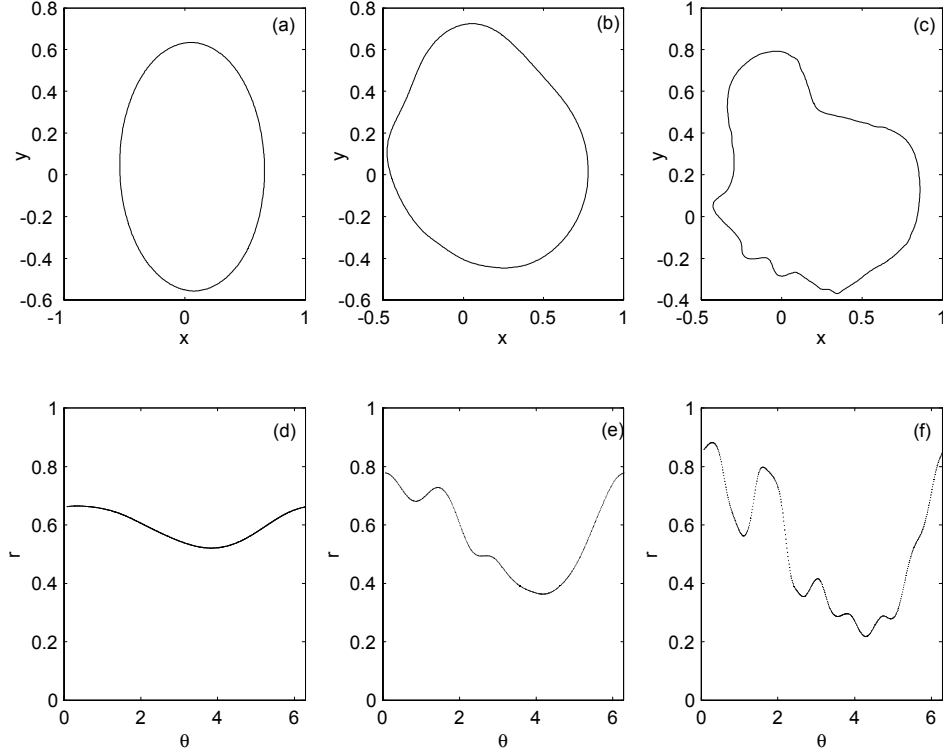


Figure 4.1: Continuation of invariant curve of the map (4.1) (a,b,c) in the 2-D cartesian coordinate and (d,e,f) are in the polar coordinate equation (4.3) (taking θ as the parameter) for $\lambda = 2.5$, $\omega = 0.8$ and (a) $\epsilon = 0.1$ (b) $\epsilon = 0.25$ (c) $\epsilon = 0.315$ (d) $\epsilon = 0.1$ (e) $\epsilon = 0.25$ (f) $\epsilon = 0.315$

This map $\mathbf{f}$ has m as the system parameter on which continuation can be performed. The above system can be numerically simulated to obtain the invariant curve, say for $m = 0.0$. This invariant curve is sampled at 20 equally spaced points $\mathbf{x}_k^{(0)}$. The family of event surfaces are chosen in the same manner as in the previous example. The system of equations 3.3 are formulated and the Newton-Raphson method is used to locate the invariant tori (figure 4.2). Again, as in the previous example resampling of the data points is done and the Newton-Raphson formulation is re-run in the hope that one is closer to the actual invariant curve.

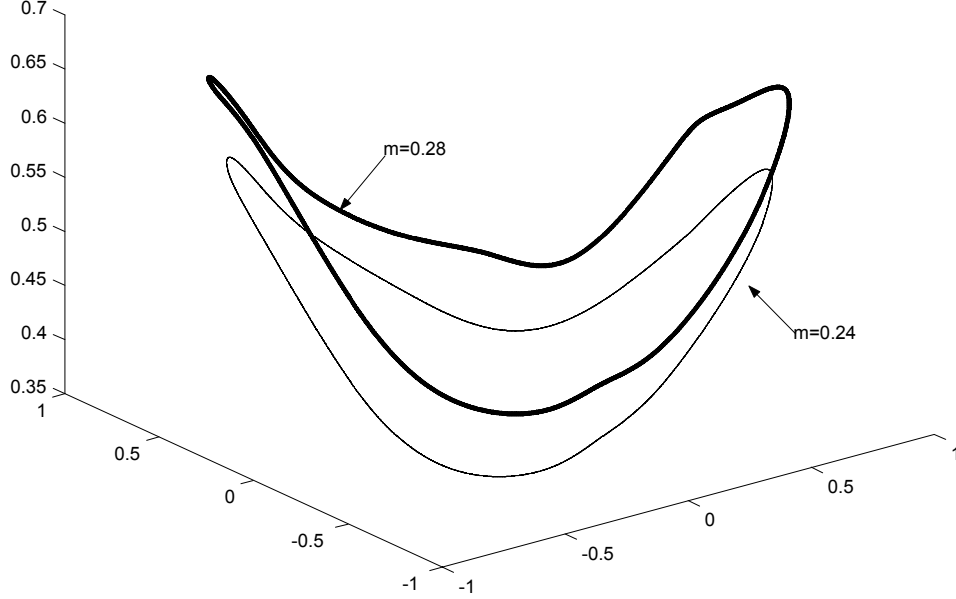


Figure 4.2: Continuation of the invariant curve for the flow map of equation (4.5) in 3-D state space for $m = 0.24$ and $m = 0.28$

4.2 Two-Dimensional Torus

Consider a discrete dynamical system given by the map

$$\mathbf{f}(\mathbf{x}) = \phi(\mathbf{x}, 1) \quad (4.6)$$

where $\phi(\mathbf{x}, 1)$ is the *flow* function of the following continuous dynamical system

$$\begin{aligned} \dot{x}_1 &= \frac{x_1 x_3}{a + x_1} - x_2 \omega \\ \dot{x}_2 &= \frac{x_2 x_3}{a + x_1} + x_1 \omega \\ \dot{x}_3 &= c x_3 (1 - x_1^2) - x_1 \end{aligned} \quad (4.7)$$

where

$$c = \alpha \frac{x_1^2}{a + x_1} + \beta \quad x_1 = \sqrt{x_1^2 + x_2^2} - a$$

The above map has a *donut* shaped invariant torus, which can be parameterized by two variables t and s . The initial guess points $\mathbf{x}_{k_1, k_2}^{(0)}$, ($k_1 = 1, \dots, N$, $k_2 = 1, \dots, M$) are sampled

on the torus given by:

$$\begin{aligned} x &= (5 + 2\cos\theta) \cos\phi \\ y &= (5 + 2\cos\theta) \sin\phi \end{aligned} \quad (4.8)$$

$$z = 2\sin\theta \quad (4.9)$$

These sampled points are used as an initial guess for the system with $\omega = 5.3$, $a = 5$, $\alpha = 0.01$, $\beta = 0$. A periodic bicubic spline $\mathbf{X}(t, s; \mathbf{x}_1^{(0)}, \dots, \mathbf{x}_N^{(0)})$ is used to interpolate a surface passing through the sampled points. The point on the interpolant closest to the image $\mathbf{f}(\mathbf{x}_{k_1, k_2}^{(0)})$ is used to compute the shift $\tau_{k_1, k_2}^{(0)}(t, s)$ and $\sigma_{k_1, k_2}^{(0)}(t, s)$ for each point.

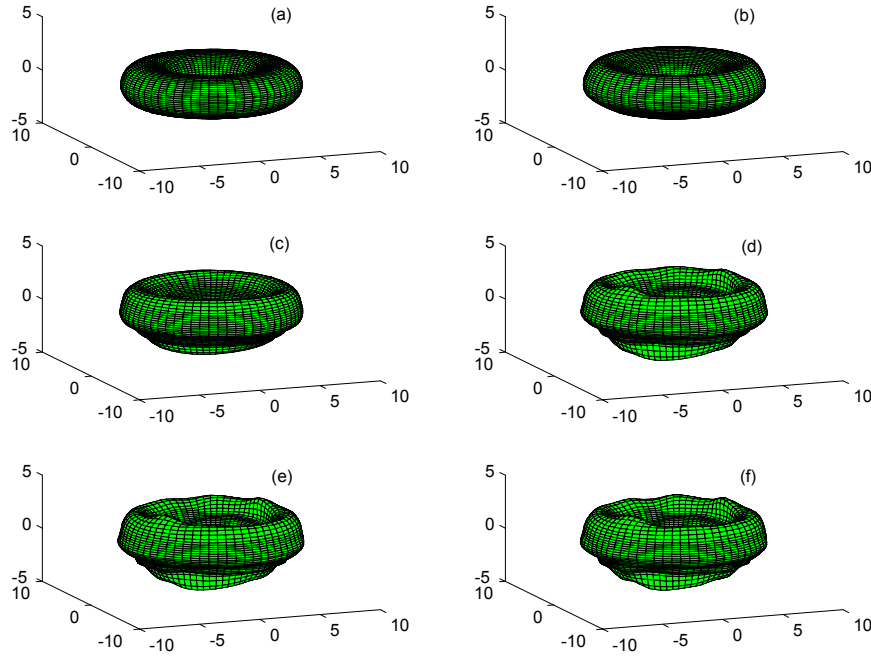


Figure 4.3: Continuation of invariant torus in 3-D space for the flow map of equation (4.7) for $\omega = 5.3$, $a = 5$ and (a) $\alpha = 0$, $\beta = 0$ (b) $\alpha = 0$, $\beta = 0.4$ (c) $\alpha = 1.0$, $\beta = 0.4$ (d) $\alpha = 1.5$, $\beta = 0.4$ (e) $\alpha = 1.51$, $\beta = 0.4$ (f) $\alpha = 1.52$, $\beta = 0.4$

The families of event functions $h_{k_1, k_2}^1(\mathbf{x})$ and $h_{k_1, k_2}^2(\mathbf{x})$ are chosen as

$$\begin{aligned} h_{k_1, k_2}^1(\mathbf{x}) &= \left(\mathbf{x} - \mathbf{x}_{k_1, k_2}^{(0)} \right) \bullet \frac{\partial \mathbf{X}}{\partial t} \left(t_{k_1, k_2}, s_{k_1, k_2}; \mathbf{x}_{1,1}^{(0)}, \dots, \mathbf{x}_{N,M}^{(0)} \right) \\ h_{k_1, k_2}^2(\mathbf{x}) &= \left(\mathbf{x} - \mathbf{x}_{k_1, k_2}^{(0)} \right) \bullet \frac{\partial \mathbf{X}}{\partial s} \left(t_{k_1, k_2}, s_{k_1, k_2}; \mathbf{x}_{1,1}^{(0)}, \dots, \mathbf{x}_{N,M}^{(0)} \right) \end{aligned}$$

Once the initial guess points $\mathbf{x}_{k_1,k_2}^{(0)}(t,s)$, $\tau_{k_1,k_2}^{(0)}(t,s)$, $\sigma_{k_1,k_2}^{(0)}(t,s)$ and the event surfaces $h_{k_1,k_2}^1(\mathbf{x},t,s)$ and $h_{k_1,k_2}^2(\mathbf{x},t,s)$ are chosen, the system of equations 3.7a, 3.7b, 3.7c are formulated and the Newton-Raphson method is used to locate the invariant torus.

For the results shown in fig. 4.3, the torus is sampled by taking 21 evenly spaced points in each of the parameter directions. For nonzero values of alpha cross-section of the torus changes as one takes different angular sections figure 4.4. Continuation of the invariant torus is done by first increasing β and subsequently increasing α .

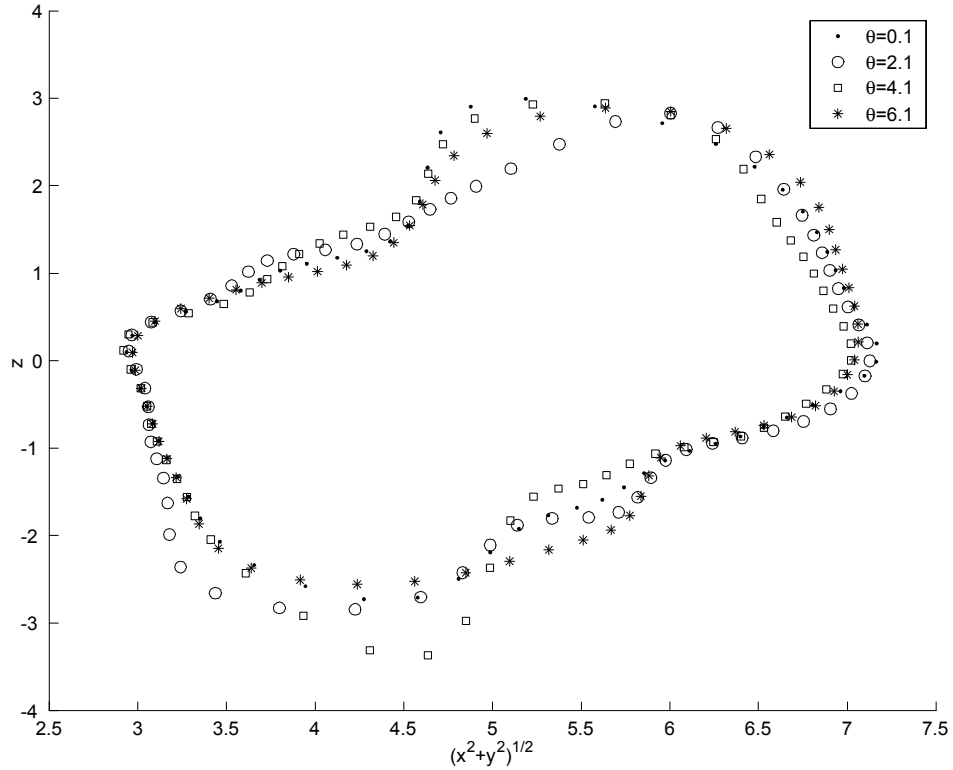


Figure 4.4: Different cross sections of the invariant torus ($\alpha = 1.52, \beta = 0.4, a = 5.0, \omega = 5.3$)

Once we have located the invariant torus, as a simple geometric application one can compute the principal curvatures and there directions at any given point on the torus. Figure 4.5 shows the principal directions and the normal vector at a point corresponding to $t = 1.0$ and $s = 1.0$ having principal curvature $\kappa_1 = -0.15297$ and $\kappa_2 = -0.9464$ (section 2.5.2).

Next let us consider the example of a forced van-der Pol oscillator.

$$\ddot{x} + \nu(x^2 - 1)\dot{x} + x = \epsilon F \cos(\omega t)$$

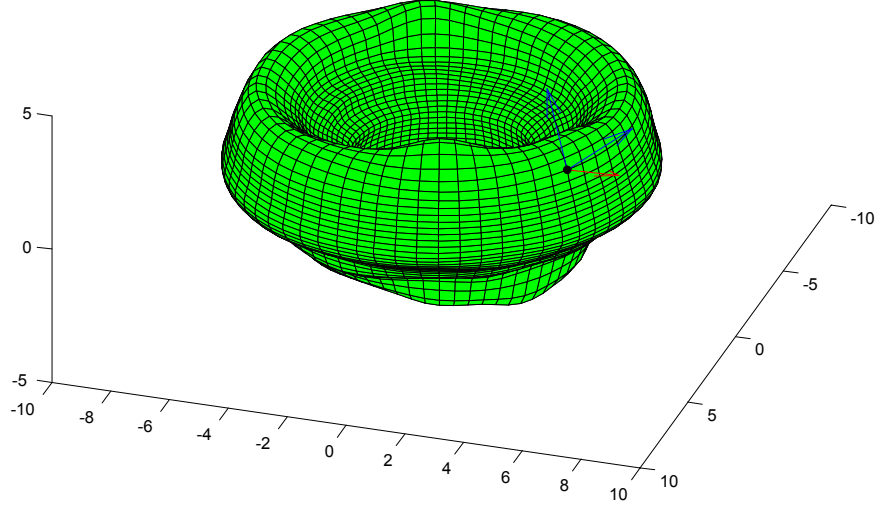


Figure 4.5: The principal curvature directions and the normal at a point corresponding to $t = 1.0$, $s = 1.0$ on the invariant torus ($\alpha = 1.52, \beta = 0.4, a = 5.0, \omega = 5.3$)

This is a non-autonomous system, which can be treated as an autonomous system by introducing the phase variable as a state variable:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\nu(x_1^2 - 1)x_2 - x_1 + \epsilon F \cos(x_3) \\ \dot{x}_3 &= \omega \end{aligned} \tag{4.10}$$

This autonomous system has a solution that lies on a *topological* cylinder $S^1 \times R^1$ which extends from negative infinity to positive infinity in the x_3 direction. However, this cylinder is periodic i.e. given any cross section $x_3 = x_0$ it repeats after $\Delta x_3 = 2\pi$, thus obtaining the characteristic torus in three-dimensional space.

Consider the discrete dynamical associated with the above continuous system

$$\mathbf{f}(\mathbf{x}) = \phi(\mathbf{x}, 1)$$

where $\phi(\mathbf{x}, 1)$ is the *flow* function of the above continuous. Choosing x_3 as one of the parameters parametrizing the torus, reduces the number of unknowns as the shift in that

parameter direction is known. Using the formulation proposed in section 3.2 one can solve for the unknowns (the sampled points and the shift).

For the results shown in figure 4.6, a straight cylinder with 16 sections (in the x_3 direction) each having 34 equally spaced points is taken as an initial guess. The cross-section of the cylinder is the invariant periodic curve for the unforced system. The system parameters taken for the initial guess are for $F = 1$, $\omega = 1.01$, $\epsilon = 0$ and $\nu = 0.01$. The initial guess for the shift and the event surfaces are chosen in the same manner as in the previous example. Continuation can be done by varying the amplitude of the periodic forcing. This can be achieved by varying ϵ .

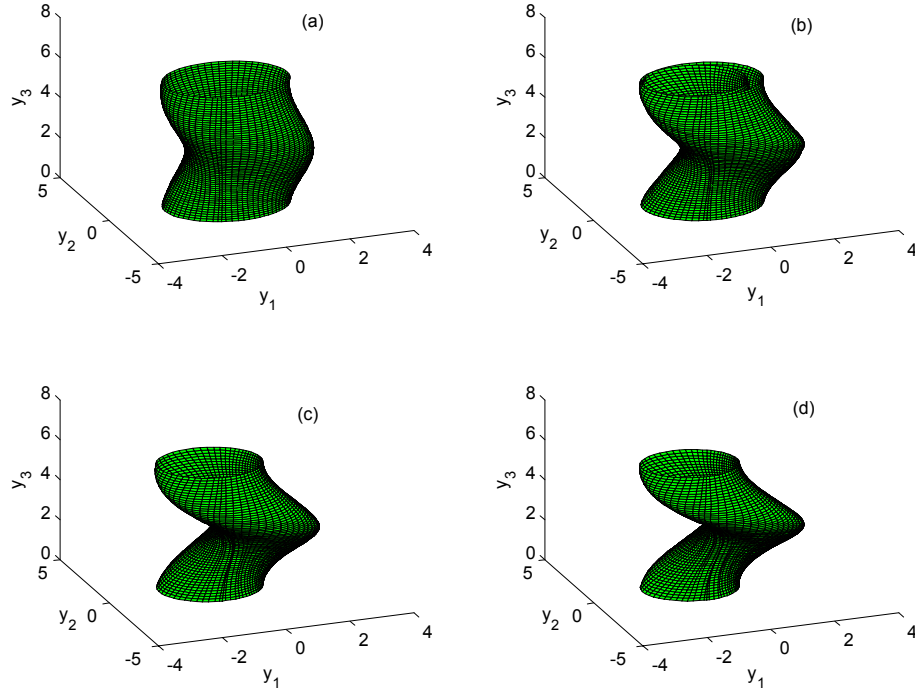


Figure 4.6: The characteristic topological cylinder for the van-der Pol oscillator for $F = 1$, $\nu = 0.01$, $\omega = 1.01$ and (a) $\epsilon = 0.01$ (b) $\epsilon = 0.015$ (c) $\epsilon = 0.02$ (d) $\epsilon = 0.022$

Next, consider equation 4.5 with periodic forcing. This is a non-autonomous system, which

can be treated as an autonomous system by taking the phase as a state variable:

$$\begin{aligned}
\dot{x}_1 &= mx_1 - x_2 - x_1x_3 \\
\dot{x}_2 &= mx_2 + x_1 + A\cos(x_4) \\
\dot{x}_3 &= -x_3 + x_2^2 + x_1^2x_3 \\
\dot{x}_4 &= \omega
\end{aligned} \tag{4.11}$$

This autonomous system has a solution that lies on a *topological* cylinder $S^1 \times R^2$ which extends from negative infinity to positive infinity in the x_4 direction. However, this cylinder is periodic i.e. given any cross section $x_4 = x_0$ it repeats after $\Delta x_4 = 2\pi$, thus one obtains the characteristic torus in three-dimensional space.

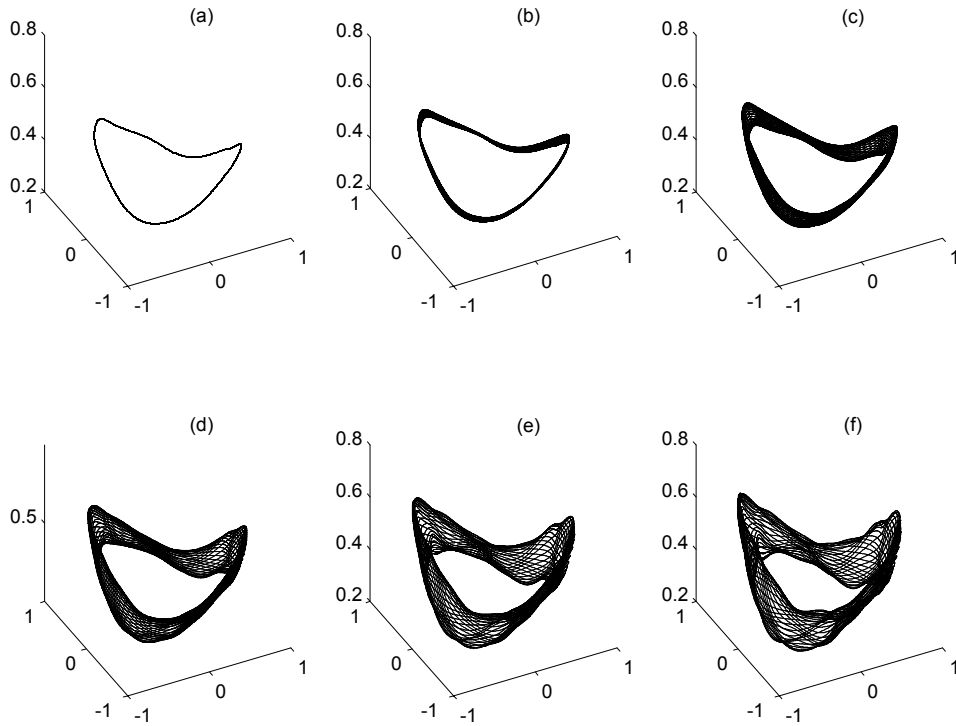


Figure 4.7: The projection of the characteristic topological cylinder in 4-D state space in x_1 - x_2 - x_3 state space for the equation 4.11 obtained by continuation for $\omega = 1.9, m = 0.21$ and (a) $F = 0.0$ (b) $F = 0.01$ (c) $F = 0.03$ (d) $F = 0.05$ (e) $F = 0.06$ (f) $F = 0.08$

Consider the discrete dynamical associated with the above continuous system

$$\mathbf{f}(\mathbf{x}) = \phi(\mathbf{x}, 1)$$

where $\phi(\mathbf{x}, 1)$ is the *flow* function of the above continuous. Choosing x_4 as one of the parameters parametrizing the torus, reduces the number of unknowns as the shift in that parameter direction is known. Using the formulation proposed in section 3.2 one can solve for the unknowns (the sampled points and the shifts). The system parameters for this system are F and m . For the results shown in figure 4.7, a straight cylinder with 30 sections (in the x_4 direction) each having 20 equally spaced points is taken as an initial guess. The cross-section of the cylinder is the invariant periodic curve for the unforced system. The system parameters taken for the initial guess are for $m = 0.21$, $\omega = 1.9$ and $F = 0$. The initial guess for the shifts and the event surfaces are chosen in the same manner as in the previous example. Continuation can be done by varying the amplitude of the periodic forcing. This can be achieved by varying F .

4.3 Sweeping along an n -dimensional torus

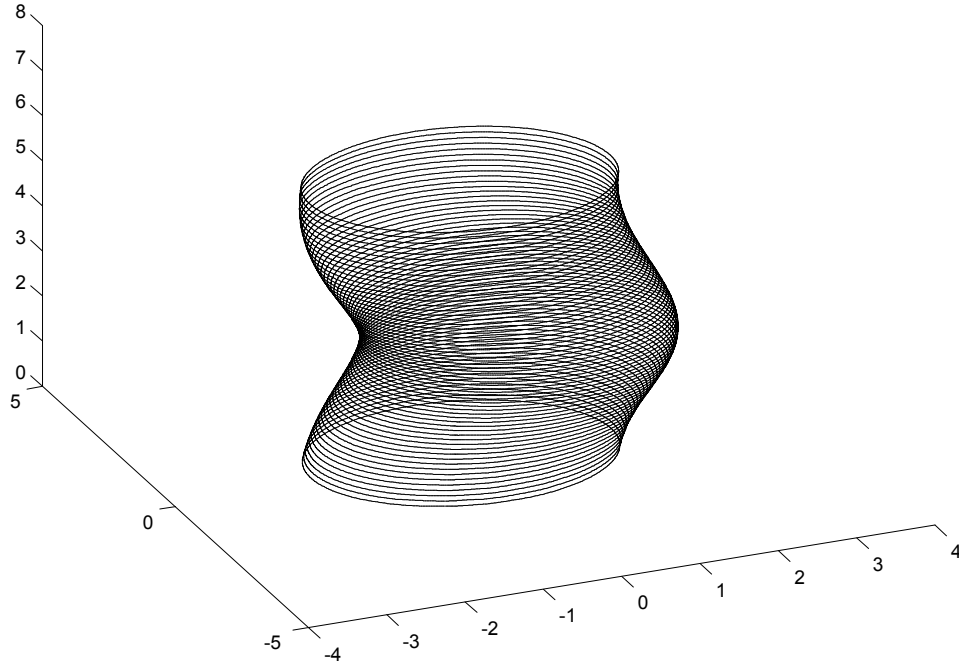


Figure 4.8: The characteristic topological cylinder for the van-der Pol oscillator obtained by the sweeping technique, $F = 1$, $\nu = 0.01$, $\epsilon = 0.01$, $\omega = 1.01$

In the case of the forced van-der-Pol oscillator in the previous section, the invariant tori

intersect constant- x_3 planes in closed curves that are invariant under the map

$$\mathbf{f}(\mathbf{x}) = \mathbf{P}(x), \quad (4.12)$$

where $\mathbf{P}$ is the Poincaré map associated with the Poincaré section given by the corresponding constant- x_3 plane. One may thus employ the methodology for one-dimensional tori in each such section. Indeed, since the torus is continuous, a converged solution in one such section may be used (possibly after some resampling as described previously) as an initial guess for the invariant curve in a nearby section. In this way, the value of x_3 in each section is used as the continuation parameter. In this fashion, a family of one-dimensional tori sweeps along the two-dimensional torus. Figure 4.8 shows the result of applying such a sweeping technique to the forced van-der-Pol oscillator.

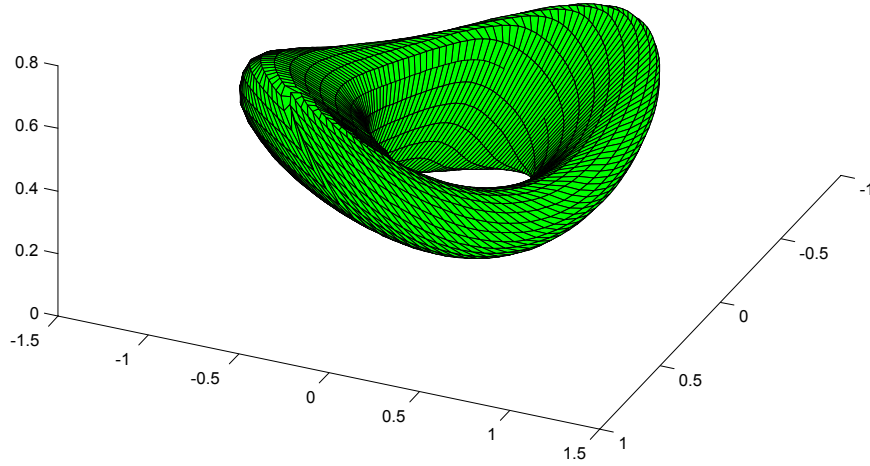


Figure 4.9: The projection of the characteristic topological cylinder in 4-D state space in x_1 - x_2 - x_3 state space for the equation 4.11 obtained by the sweeping technique, $F = 1$, $\omega = 1.9$, $m = 0.21$

One can also compute the characteristic torus for the system given by equation 4.11 by using the sweep technique. Here, the invariant tori intersect constant- x_4 planes in closed curves that are invariant under the map

$$\mathbf{f}(\mathbf{x}) = \mathbf{P}(x), \quad (4.13)$$

where $\mathbf{P}$ is the Poincaré map associated with the Poincaré section given by the corresponding constant- x_4 plane. As described earlier, one can then sweep across in the x_4 direction from $x_4 = x_0$ to $x_4 = x_0 + 2\pi$ to obtain the characteristic torus figure 4.9.

Chapter 5

Conclusion

In this thesis a new method has been proposed for the evaluation of invariant tori of maps. The approach of evaluating the invariant tori of maps requires one to find a torus function $u : T^p \rightarrow R^n$ (T^p is the p -dimensional torus) such that its image $K : u(\theta) | \theta \in T^p$ is invariant under the map $\mathbf{F} : R^n \rightarrow R^n$. That is, the invariance condition

$$u(\mathcal{T}(\theta)) = \mathbf{F}(u(\theta)) \quad (5.1)$$

holds pointwise, where $\theta \in T^p$ and $\mathcal{T} : T^p \rightarrow T^p$ is diffeomorphic to the map $\mathbf{F}$ restricted to the invariant torus K . The above equation provides only an equation in u , but the function $\mathcal{T}$ is also unknown and depends on the parameterization of the torus K . Once $\mathcal{T}$ is fixed one can solve for the torus. The proposed method formulates an auxiliary equation whereby the function $\mathcal{T}$ can be computed for any n -dimensional problem.

The method proposed requires a good initial guess of the invariant torus. The points on this torus should be sampled in such a manner that one has a periodic set of sample points in each parameter direction. These points should also almost be equi-distant both on the torus and in the parameter space. As continuation is done on the torus, one may require to resample the points on the computed invariant torus to ensure that the points are fairly well distributed.

A n -dimensional tori can be parameterized by n parameters. The sampled points on the torus are picked up in each of these parameter directions such that they are *periodic*. Since these parameter directions are independent, one can interpolate a function for each coordinate point in each parameter direction independently. Hence, obtaining an interpolated hyper-surface passing through the sampled points.

The interpolation function, used in the present work are the cubic splines and the fourier interpolant. The choice of fourier interpolant requires that we sample even number of equally spaced points. The interpolant obtained has C^∞ continuity. The constrain of using even number of points and evenly spaced points is relaxed with cubic splines. However

the interpolant has only C^3 continuity. Cubic splines has been used for two and higher dimensional torus. This choice of interpolation function requires that the sampled point are ordered in the increasing parameter direction. These points should not be clustered in some region of the state space, in other words these points should be distributed almost evenly. Clustering causes the interpolant to be oscillatory which may cause the system of equation either not to converge or may have a large deviation form the true solution. The clustering that happens due to the dynamics of the system is removed by redistributing the sampled points. The redistribution of points on the converged torus also ensures that the obtained solution is close to the true solution. Redistribution is done by picking up equidistant points along the interpolant in each of the parameter directions. This technique is very helpful for one-dimensional tori but needs to be improved for higher dimensional tori.

The method proposed has been demonstrated to compute invariant curves in two and three-dimensional state space and for two-dimensional torus in three and four dimensional state space. The code written to compute the two dimensional torus can be used to compute the 2-torus in a n-dimensional state space.

Bibliography

- [1] Yuri A. Kuznetsov, *Elements of Applied Bifurcation Theory*, Springer-Verlag New York, 1998.
- [2] Chr. Kaas-Petersen, *Computation of quasiperiodic solutions of forced dissipative systems I*, J.Comput. Phys., 58(3)(1985), pp.395-408.
- [3] Chr. Kaas-Petersen, *Computation of quasiperiodic solutions of forced dissipative systems II*, J.Comput. Phys., 64(2)(1986), pp.433-442.
- [4] Chr. Kaas-Petersen, *Computation, continuation, and bifurcation of torus solutions for dissipative maps and ordinary differential equation*, Phys. D, 25(1-3)(1987), pp.288-306.
- [5] I.G. Keverkidis, R. Aris, L.D Schmidt and S Pelikan, *Numerical Computations of invariant circles of maps*, Physica D, 16 (1985), pp. 243-251.
- [6] S. Wiggins, *Normally Hyperbolic Invariant Manifolds in Dynamical Systems*, Springer-Verlag, 1994
- [7] Ali. H Nayfeh, Balakumar Balachandran, *Applied Nonlinear Dynamics*, John Wiley & Sons, 1995
- [8] Steven H. Strogatz, *Nonlinear Dynamics and Chaos*, Westview, 1994
- [9] D.K Arrowsmith and C.M Place, *An Introduction to Dynamical Systems*, Cambridge University Press, 1990
- [10] Erwin Kreyszig, *Advanced Engineering Mathematics*, John Wiley & Sons, 1999
- [11] Erwin Kreyszig, *Differential Geometry*, Dover Publications, New York, 1991
- [12] W.H Press, S.A Teukolsky, W.T Vetterling, B.P Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992
- [13] R. H. Bartels, J.C. Beatty and B.A. Barsky, *Hermite and Cubic Spline Interpolation*, Ch. 3 in An Introduction to Splines for Use in Computer Graphics and Geometric Modelling. San Francisco, CA: Morgan Kaufmann, pp. 9-17, 1998.

- [14] W. E. Boyce and R. C. DiPrima, *Elementary Differential Equations and Boundary Value Problems*, 5th ed. New York: Wiley, 1992.
- [15] P. Lawrence, *Differential Equations and Dynamical Systems*, 2nd ed. Springer-Verlag New York, 1996
- [16] M. Berger and B. Gostiaux, *Differential Geometry: Manifolds, Curves and Surfaces*, Springer-Verlag New York, 1988
- [17] Eric W. Weisstein, "Newton's Method." *From MathWorld—A Wolfram Web Resource*, <http://mathworld.wolfram.com/NewtonsMethod.html>
- [18] A. Mees, *Dynamics of Feedback Systems*, Wiley, Chichester, England, 1981
- [19] K.S Kundert, A. Sangiovanni-Vincentelli and T. Sugawara, *Techniques for Finding the Periodic Steady-State Response of Circuits*, Marcel Dekker, New York, 1987
- [20] K. Neymeyr and F.F Seelig, *Determination of unstable limit cycles in chaotic systems by the method of unrestricted harmonic balance*, Z. Naturforsch. 46, 1991
- [21] J. Rinzel and R.N Miller, *Numerical calculation of stable and unstable periodic solutions to the Hodgkin-Huxley equation*, Math. Biosci, 1980
- [22] T.S Parker and L.O Chua, *Practical Numerical Algorithms for Chaotic Systems*, Springer-Verlag, New York, 1989
- [23] H.B Keller, *Numerical Methods for Two-Point Boundary-Value Problems*, Blaisdell, Waltham, Massachusetts, 1968
- [24] J.H Curry, Z. Nitecki and C. Robinson, *An Algorithm for finding closed orbit, in Global Theory of Dynamical Systems*, Springer-Verlag, New York, 1980
- [25] A.M Samoilenko and N.I Ronto, *Numerical-Analytical Method of Investigation Periodic Solutions*, Mir Publications, Moscow, 1979
- [26] A.M Samoilenko, *Elements of the Mathematical Theory of Multi-frequency Oscillations*, English translation, Kluwer Academic Publishers, Dordrecht et al, 1991
- [27] L.O. Chua and A. Ushida, *Algorithms for computing almost periodic steady state response of nonlinear system to multiple input frequencies*, IEEE Trans. Circuits and Systems, 28(10) (1981), pp 953-971.
- [28] M.R. Trummer, *Spectral methods in computing invariant tori*, Applied Numerical Mathematics 34 2000, pp 275-292.
- [29] G. Moore, *Computation and parameterisation of invariant curves and tori*, SIAM J. of Numerical Anal. 33(6) (1996), pp 2333-2358.

- [30] F. Schilder, H. M. Osinga, W. Vogt, *Continuation of Quasi-Periodic Invariant Tori*, www.enm.bris.ac.uk/anm/preprints/2004r11.pdf

Appendix A

Matlab Code

A.1 One-Dimensional Torus

In this section, the matlab codes used to compute the invariant one-dimensional code are given. The flowing list gives the name of the functions used

1. *run_update* :The main script that calls all other functions.
2. *guess* :Calculates the initial guess *shift* and the the tangents at every reference point
3. *Newton* :Calculates $\mathbf{F}$ and it Jacobian.
4. *processing_spline* :This function calls all the functions that calculates the Cubic splines and does various operations on it.
 - 4a. *spline_periodic_u* :Calculates the slope and the derivative matrix at each sample point.
 - 4b. *interpolate_spline* :Calculates the interpolated value at any given point on the spline.
 - 4c. *interpolate_spline_deriv_sample* :Calculates the derivative matrix at any given point on the spline.
 - 4d. *interpolate_spline_deriv_time* :Calculates the slope at any given point on the spline.
5. *processing_fft* :Calculates the Fourier interpolant, slope, derivative matrix at any given point on the interpolant.
6. *dynamical_system* :Call the function that defines the vector field.

Function: run_update

```

function [] = run_update(x,N,dim,sys_arg,bif_par,n,method,interpolation_type)

% interpolation_type
%1 is Fourier interpolation
%2 is cubic splines

%N=No of points on the curve
%dim = the dimension of the dynamical system
%sys_arg=[lambda,epsilon,omega];
%bif_par=[par_start,par_end,par_step];
%n= No of times to apply the map in the guess
%method= refers to resampling
% 1: to resample by equal arc length or by
% b) inseting point as the slope vary

close all;
L=[];
Max=[];

figure(3);
hold on;

tin=[2*pi/N:2*pi/N:2*pi];
if(length(x)==0)
    if(dim ==2)
        x=[.5*cos(tin'),.5*sin(tin')];
    else
        x=[0.5*cos(tin'),0.5*sin(tin'),0.5*sin(tin')];
    end
    [xref,tangents,tauref]=guess(x,dim,tin,sys_arg,n,N,interpolation_type);
else
    [xref,tangents,tauref]=guess(x,dim,tin,sys_arg,0,N,interpolation_type);
    % n=0 assuming the x vector is a converged solution
end

x=xref;
tau=tauref;

for par =bif_par(1):bif_par(3):bif_par(2)
    count =0;
    par
    sys_arg(1)=par;
    F=1;
    while(max(abs(F))>1e-12 & count<15)
        [F,dF]=Newton(x,dim,tau,N,xref,tin,tangents,sys_arg,interpolation_type);
        temp=[];
        for i =1:dim
            temp=[temp;x(:,i)];
        end
        temp=[temp;tau']-inv(dF)*F;
        for i=1:dim
            x(1:N,i)=temp((i-1)*N+1:i*N);
        end
        tau=temp(dim*N+1:(dim+1)*N)';
        max(abs(F))
        count=count+1;
    end

    %resampling the curve
    %M=No of equally spaces points on the curve

```

```

if(strcmp(input('do you wish to resample => ','s'),'y'))
M=input('no of points =>')
[x tout]=resample_update(x,dim,tin,N,M,method,interpolation_type);
%if we introduce new points in resampling than tin = tout(in case
%of splines only as the delta will vary and we cannot use fourier)
if(N~=M)
N=M
tin=[2*pi/M:2*pi/M:2*pi];
end

if(method==2 & interpolation_type==2)
tin=tout;
end

[xref,tangents,tauref]=guess(x,dim,tin,sys_arg,0,M,interpolation_type);
x=xref;
tau=tauref;

F=1.0;
count=0;

%check convergance
while(max(abs(F))>1e-12 & count<15)
[F,dF]=Newton(x,dim,tau,N,xref,tin,tangents,sys_arg,interpolation_type);
temp=[x(:,1);x(:,2);tau]-inv(dF)*F;
x=[temp(1:N,1) temp(N+1:2*N,1)];
tau=temp(2*N+1:3*N,1);
max(abs(F))
count = count+1;
pause
end
end
%updating

L=[L;par];
Max=[Max;max(x(1:2*N))];
save D_100 x
%Updating the U_ref and Tauref and the Tangent vector
xref=x;
tauref=tau;
if(interpolation_type==1)
if(dim==2)
xreff=[processing_fft('fft',xref(:,1)),processing_fft('fft',xref(:,2))];
tangents=[processing_fft('interpolate_deriv_time',xreff(:,1),tin,2*pi/N),...
processing_fft('interpolate_deriv_time',xreff(:,2),tin,2*pi/N)];
elseif(dim==3)
xreff=[processing_fft('fft',xref(:,1)),processing_fft('fft',xref(:,2)),...
processing_fft('fft',xref(:,3))];
tangents=[processing_fft('interpolate_deriv_time',xreff(:,1),tin,2*pi/N),...
processing_fft('interpolate_deriv_time',xreff(:,2),tin,2*pi/N),...
processing_fft('interpolate_deriv_time',xreff(:,3),tin,2*pi/N)];
end
elseif(interpolation_type==2)
if(dim==2)
m1=processing_spline('spline_periodic',tin,x(:,1));
m2=processing_spline('spline_periodic',tin,x(:,2));
tangents=[processing_spline('spline_periodic',tin,xref(:,1),m1),...
processing_spline('spline_periodic',tin,xref(:,2),m2)];
elseif(dim==3)

```

```

    m1=processing_spline('spline_periodic',tin,x(:,1));
    m2=processing_spline('spline_periodic',tin,x(:,2));
    m3=processing_spline('spline_periodic',tin,x(:,3));
    tangents=[processing_spline('spline_periodic',tin,xref(:,1),m1),...
        processing_spline('spline_periodic',tin,xref(:,2),m2),...
        processing_spline('spline_periodic',tin,xref(:,3),m3)];
end

end

figure(2)
title('Bifurcation Diagram');
xlabel('\lambda');
ylabel('X_{max}');
plot(L,Max,'.');
save bif L Max x tau

function [xref,tangents,tau]=guess(x,dim,tin,sys_arg,n,N,interpolation_type)

temp=x;
for i=1:n
    temp=dynamical_sys(temp,sys_arg);
end

xref=temp;
if(interpolation_type==1)
    if(dim==2)
        xreff=[processing_fft('fft',xref(:,1)),processing_fft('fft',xref(:,2))];
        tangents=[processing_fft('interpolate_deriv_time',xreff(:,1),tin',2*pi/N),...
            processing_fft('interpolate_deriv_time',xreff(:,2),tin',2*pi/N)];
    elseif(dim==3)
        xreff=[processing_fft('fft',xref(:,1)),processing_fft('fft',xref(:,2)),...
            processing_fft('fft',xref(:,3))];
        tangents=[processing_fft('interpolate_deriv_time',xreff(:,1),tin',2*pi/N),...
            processing_fft('interpolate_deriv_time',xreff(:,2),tin',2*pi/N),...
            processing_fft('interpolate_deriv_time',xreff(:,3),tin',2*pi/N)];
    end
elseif(interpolation_type==2)
    if(dim==2)
        m1=processing_spline('spline_periodic',tin,x(:,1)');
        m2=processing_spline('spline_periodic',tin,x(:,2)');
        tangents=[m1',m2'];
    elseif(dim==3)
        m1=processing_spline('spline_periodic',tin,x(:,1)');
        m2=processing_spline('spline_periodic',tin,x(:,2)');
        m3=processing_spline('spline_periodic',tin,x(:,3)');
        tangents=[m1',m2',m3'];
    end
end

tout=[0:.001:2*pi]';
if(interpolation_type==1)
    if(dim==2)
        xi=[processing_fft('interpolate',xreff(:,1),tout,2*pi/N),...
            processing_fft('interpolate',xreff(:,2),tout,2*pi/N)];
    elseif(dim==3)
        xi=[processing_fft('interpolate',xreff(:,1),tout,2*pi/N),...
            processing_fft('interpolate',xreff(:,2),tout,2*pi/N),...
            processing_fft('interpolate',xreff(:,3),tout,2*pi/N)];
    end
end

```

```

elseif(interpolation_type==2)
    if(dim==2)
        xi=[processing_spline('interpolate',tin,xref(:,1)',tout,m1)',...
            processing_spline('interpolate',tin,xref(:,2)',tout,m2)'];
    elseif(dim==3)
        xi=[processing_spline('interpolate',tin,xref(:,1)',tout,m1)',...
            processing_spline('interpolate',tin,xref(:,2)',tout,m2)',...
            processing_spline('interpolate',tin,xref(:,3)',tout,m3)'];
    end

end

y = dynamical_sys(xref,sys_arg);
N=length(tin);
tau=zeros(1,N);
for i=1:N
    temp=ones(length(tout),1)*y(i,:)-xi;
    if(dim==2)
        [temp,minj]=min(sqrt(temp(:,1).^2+temp(:,2).^2));
    elseif(dim==3)
        [temp,minj]=min(sqrt(temp(:,1).^2+temp(:,2).^2+temp(:,3).^2));
    end
    tau(i)=mod(tout(minj)-tin(i),2*pi);
end

```

Function: Newton

```

function [F,dF]=Newton(x,dim,tau,N,xref,tin,tangents,sys_arg,interpolation_type)

if(interpolation_type ==1)

    if(dim==2)
        xf=[processing_fft('fft',x(:,1)),processing_fft('fft',x(:,2))];
    elseif(dim==3)
        xf=[processing_fft('fft',x(:,1)),processing_fft('fft',x(:,2)),...
            processing_fft('fft',x(:,3))];
    end

    [y,dy]=dynamical_sys(x,sys_arg);

    if(dim==2)
        F = [y(:,1)-processing_fft('interpolate',xf(:,1),(tin+tau)',2*pi/N);...
            y(:,2)-processing_fft('interpolate',xf(:,2),(tin+tau)',2*pi/N);...
            (x(:,1)-xref(:,1)).*tangents(:,1)+...
            (x(:,2)-xref(:,2)).*tangents(:,2)];

        dF=zeros(3*N,3*N);
        dF(1:N,1:N)=diag(dy(:,1))-
            processing_fft('interpolate_deriv_sample',xf(:,1),(tin+tau)',2*pi/N);
        dF(1:N,N+1:2*N)=diag(dy(:,3));
        dF(1:N,2*N+1:3*N)=-diag(processing_fft('interpolate_deriv_time',xf(:,1),
            (tin+tau)',2*pi/N));
        dF(N+1:2*N,1:N)=diag(dy(:,2));
        dF(N+1:2*N,N+1:2*N)=diag(dy(:,4))-
            processing_fft('interpolate_deriv_sample',xf(:,2),(tin+tau)',2*pi/N);
        dF(N+1:2*N,2*N+1:3*N)=-diag(processing_fft('interpolate_deriv_time',
            xf(:,2),(tin+tau)',2*pi/N));
        dF(2*N+1:3*N,1:N)=diag(tangents(:,1));
        dF(2*N+1:3*N,N+1:2*N)=diag(tangents(:,2));
    elseif(dim==3)

```

```

F = [y(:,1)-processing_fft('interpolate',xf(:,1),tin+tau,2*pi/N);...
     y(:,2)-processing_fft('interpolate',xf(:,2),tin+tau,2*pi/N);...
     y(:,3)-processing_fft('interpolate',xf(:,3),tin+tau,2*pi/N);...
     (x(:,1)-xref(:,1)).*tangents(:,1)+...
     (x(:,2)-xref(:,2)).*tangents(:,2)+...
     (x(:,3)-xref(:,3)).*tangents(:,3)];

dF=zeros(4*N,4*N);

dF(1:N,1:N)=diag(dy(:,1))-processing_fft('interpolate_deriv_sample',
     xf(:,1),tin+tau,2*pi/N);
dF(1:N,N+1:2*N)=diag(dy(:,4));
dF(1:N,2*N+1:3*N)=diag(dy(:,7));
dF(1:N,3*N+1:4*N)=-diag(processing_fft('interpolate_deriv_time',
     xf(:,1),tin+tau,2*pi/N));

dF(N+1:2*N,1:N)=diag(dy(:,2));
dF(N+1:2*N,N+1:2*N)=diag(dy(:,5))-
     processing_fft('interpolate_deriv_sample',xf(:,2),tin+tau,2*pi/N);
dF(N+1:2*N,2*N+1:3*N)=diag(dy(:,8));
dF(N+1:2*N,3*N+1:4*N)=-diag(processing_fft('interpolate_deriv_time',
     xf(:,2),tin+tau,2*pi/N));

dF(2*N+1:3*N,1:N)=diag(dy(:,3));
dF(2*N+1:3*N,N+1:2*N)=diag(dy(:,6));
dF(2*N+1:3*N,2*N+1:3*N)=diag(dy(:,9))
     -processing_fft('interpolate_deriv_sample',xf(:,3),tin+tau,2*pi/N);
dF(2*N+1:3*N,3*N+1:4*N)=-diag(processing_fft('interpolate_deriv_time',
     xf(:,3),tin+tau,2*pi/N));

dF(3*N+1:4*N,1:N)=diag(tangents(:,1));
dF(3*N+1:4*N,N+1:2*N)=diag(tangents(:,2));
dF(3*N+1:4*N,2*N+1:3*N)=diag(tangents(:,3));

end

elseif(interpolation_type ==2)

[y,dy]=dynamical_sys(x,sys_arg);

if(dim==2)
m1=processing_spline('spline_periodic',tin,x(:,1)');
m2=processing_spline('spline_periodic',tin,x(:,2)');

F = [y(:,1)-processing_spline('interpolate',tin,x(:,1)',tin+tau,m1)';...
     y(:,2)-processing_spline('interpolate',tin,x(:,2)',tin+tau,m2)';...
     (x(:,1)-xref(:,1)).*tangents(:,1)+...
     (x(:,2)-xref(:,2)).*tangents(:,2)];

dF=zeros(3*N,3*N);

[m,m_x]=processing_spline('spline_periodic',tin,x(:,1)');
dF(1:N,1:N)=diag(dy(:,1))-
     processing_spline('interpolate_deriv_sample',tin,x(:,1)',tin+tau,m_x);
dF(1:N,N+1:2*N)=diag(dy(:,3));
dF(1:N,2*N+1:3*N)=-diag(processing_spline('interpolate_deriv_time',tin,
     x(:,1)',tin+tau,m));

[m,m_x]=processing_spline('spline_periodic',tin,x(:,2)');

```

```

dF(N+1:2*N,1:N)=diag(dy(:,2));
dF(N+1:2*N,N+1:2*N)=diag(dy(:,4))-
processing_spline('interpolate_deriv_sample',tin,x(:,2)',tin+tau,m_x);
dF(N+1:2*N,2*N+1:3*N)=-diag(processing_spline('interpolate_deriv_time',
tin,x(:,2)',tin+tau,m));

dF(2*N+1:3*N,1:N)=diag(tangents(:,1));
dF(2*N+1:3*N,N+1:2*N)=diag(tangents(:,2));

elseif(dim==3)
m1=processing_spline('spline_periodic',tin,x(:,1)');
m2=processing_spline('spline_periodic',tin,x(:,2)');
m3=processing_spline('spline_periodic',tin,x(:,3)');

F = [y(:,1)-processing_spline('interpolate',tin,x(:,1)',tin+tau,m1)';...
y(:,2)-processing_spline('interpolate',tin,x(:,2)',tin+tau,m2)';...
y(:,3)-processing_spline('interpolate',tin,x(:,3)',tin+tau,m3)';...
(x(:,1)-xref(:,1)).*tangents(:,1)+...
(x(:,2)-xref(:,2)).*tangents(:,2)+...
(x(:,3)-xref(:,3)).*tangents(:,3)];

dF=zeros(4*N,4*N);

[m,m_x]=processing_spline('spline_periodic',tin,x(:,1)');
dF(1:N,1:N)=diag(dy(:,1))-
processing_spline('interpolate_deriv_sample',tin,x(:,1)',tin+tau,m_x);
dF(1:N,N+1:2*N)=diag(dy(:,4));
dF(1:N,2*N+1:3*N)=diag(dy(:,7));
dF(1:N,3*N+1:4*N)=-diag(processing_spline('interpolate_deriv_time',tin,
x(:,1)',tin+tau,m));

[m,m_x]=processing_spline('spline_periodic',tin,x(:,2)');
dF(N+1:2*N,1:N)=diag(dy(:,2));
dF(N+1:2*N,N+1:2*N)=diag(dy(:,5))-
processing_spline('interpolate_deriv_sample',tin,x(:,2)',tin+tau,m_x);
dF(N+1:2*N,2*N+1:3*N)=diag(dy(:,8));
dF(N+1:2*N,3*N+1:4*N)=-diag(processing_spline('interpolate_deriv_time',
tin,x(:,2)',tin+tau,m));

[m,m_x]=processing_spline('spline_periodic',tin,x(:,3)');
dF(2*N+1:3*N,1:N)=diag(dy(:,3));
dF(2*N+1:3*N,N+1:2*N)=diag(dy(:,6));
dF(2*N+1:3*N,2*N+1:3*N)=diag(dy(:,9))-
processing_spline('interpolate_deriv_sample',tin,x(:,3)',tin+tau,m_x);
dF(2*N+1:3*N,3*N+1:4*N)=-diag(processing_spline('interpolate_deriv_time',
tin,x(:,3)',tin+tau,m));

dF(3*N+1:4*N,1:N)=diag(tangents(:,1));
dF(3*N+1:4*N,N+1:2*N)=diag(tangents(:,2));
dF(3*N+1:4*N,2*N+1:3*N)=diag(tangents(:,3));
end
end

```

Function: processing_spline

```

function varargout=processing_spline(flag,tin,xin,varargin)
% sample point refer to the points
% time refers to the t parameter
%m slope w.r.t t

```

```

% m_x slopr w.r.t sample point data

switch flag
case 'spline_periodic'
    [m,dm]=spline_periodic_u(tin,xin);
    varargout{1}=m;
    varargout{2}=dm;
case 'interpolate'
    tout=varargin{1};
    m=varargin{2};
    varargout{1} = interpolate_spline(tin,xin,m,tout);
case 'interpolate_deriv_sample'
    Tout=varargin{1};
    dm=varargin{2};
    varargout{1}=interpolate_spline_deriv_sample(tin,xin,dm,Tout);
case 'interpolate_deriv_time'
    Tout=varargin{1};
    m=varargin{2};
    varargout{1}=interpolate_spline_deriv_time(tin,xin,m,Tout);
end

```

Function: spline_periodic_u

```

function [m,dm] =spline_periodic_u(t,x)
%return a row vector and the jacobian
%takes two row vectors

N=length(t);
dt=diff([0,t,t(1)+t(end)]);
dx=diff([x(N),x,x(1)]);
rhs(1:N)=6*(dx(2:N+1)./dt(2:N+1).^2+dx(1:N)./dt(1:N).^2);
coeff=sparse(1:N,1:N,4./dt(1:N)+4./dt(2:N+1),N,N)+...
    sparse([N,1:N-1],1:N,2./dt(1:N),N,N)+...
    sparse(1:N,[N,1:N-1],2./dt(2:N+1),N,N);
drhs=sparse(1:N,1:N,6./dt(1:N).^2-6./dt(2:N+1).^2,N,N)+...
    sparse([N,1:N-1],1:N,6./dt(1:N).^2,N,N)+...
    sparse(1:N,[N,1:N-1],-6./dt(2:N+1).^2,N,N);

m=(rhs/coeff);
dm=(drhs/coeff);

```

Function: interpolate_spline

```

function xout=interpolate_spline(tin,xin,m,tout)

s=[0 tin];
y=[xin(end) xin];
dy=[m(end) m];
xout=zeros(1,length(tout));
for i=1:length(tout)
    k=max(find(s<=mod(tout(i),2*pi)));
    t = (mod(tout(i),2*pi)-s(k))/(s(k+1)-s(k));
    p1 = (t-1)^2*(1+2*t); q1 = t*(t-1)^2;
    p2 = t^2*(3-2*t); q2 = t^2*(t-1);
    xout(i)=y(k)*p1+y(k+1)*p2+(s(k+1)-s(k))*(dy(k)*q1+dy(k+1)*q2);
end

```

Function: interpolate_spline_deriv_sample

```

function dxout=interpolate_spline_deriv_sample(tin,xin,dm,tout)

N=length(tin);
s=[0 tin];

y_d=sparse([1:N+1],[N,1:N],1,N+1,N);
dxout=zeros(length(tout),N);
dy_d=[dm(end,:);dm];%making it cyclic
for i=1:length(tout)
    k=max(find(s<=mod(tout(i),2*pi)));
    t = (mod(tout(i),2*pi)-s(k))/(s(k+1)-s(k));
    p1 = (t-1)^2*(1+2*t); q1 = t*(t-1)^2;
    p2 = t^2*(3-2*t); q2 = t^2*(t-1);
    dxout(i,:)=y_d(k,:)*p1+y_d(k+1,:)*p2+(s(k+1)-s(k))*(dy_d(k,:)*q1+dy_d(k+1,:)*q2);
end

```

Function: interpolate_spline_deriv_time

```

function xout=interpolate_spline_deriv_time(tin,xin,m,tout)

s=[0 tin];
y=[xin(end) xin];
dy=[m(end) m];
xout=zeros(1,length(tout));
for i=1:length(tout)
    k=max(find(s<=mod(tout(i),2*pi)));
    t = (mod(tout(i),2*pi)-s(k))/(s(k+1)-s(k));
    p1t = 6*t*(t-1); q1t = (3*t-1)*(t-1);
    p2t = -6*t*(t-1); q2t = t*(3*t-2);
    xout(i)=(y(k)*p1t+y(k+1)*p2t)/(s(k+1)-s(k))+dy(k)*q1t+dy(k+1)*q2t;
end

```

Function: processing_fft

```

function [H] = processing_fft(flag,varargin)
% tin should be column
switch flag
case 'fft'
    h=varargin{1};
    N=length(h);
    if(mod(N,2)~=0)
        error('The number of point should be even');
    end
    H=zeros(N,1);
    for n = 1:N
        for k =1:N
            H(n)=H(n)+h(k)*exp(2*pi*sqrt(-1)*k*n/N);
        end
    end
case 'ifft'
    h=varargin{1};
    t=varargin{2};
    delta=varargin{3};

    N=length(h);
    if(mod(N,2)~=0)
        error('The number of point should be even');
    end

```

```

end
H=zeros(N,1);
for n = 1:N
    for k =1:N
        H(n)=H(n)+h(k)*exp(-2*pi*sqrt(-1)*k*n/N);
    end
    H(n)=1/N*H(n);
end

case 'interpolate'
h=varargin{1};
t=varargin{2};
delta=varargin{3};

N= length(h);
if(mod(N,2) ~=0)
    error('The number of point should be even');
end
summa=zeros(length(t),1);
for n=1:N
    summa(:,1)=summa(:,1)+h(n)*exp(-2*pi*sqrt(-1)*(n-N*(n>N/2))/N/delta*t);
end
H=real(1/N*summa);

case 'interpolate_deriv_time'
h=varargin{1};
t=varargin{2};
delta=varargin{3};

N= length(h);
if(mod(N,2) ~=0)
    error('The number of point should be even');
end
summa=zeros(length(t),1);
for n=1:N
    summa(:,1)=summa(:,1)+(-2*pi*sqrt(-1)*(n-N*(n>N/2))/N/delta)*
    h(n)*exp(-2*pi*sqrt(-1)*(n-N*(n>N/2))/N/delta*t);
end
H=real(1/N*summa);

case 'interpolate_deriv_sample'

h=varargin{1};
t=varargin{2};
delta=varargin{3};
N= length(h);
if(mod(N,2) ~=0)
    error('The number of point should be even');
end
summa=zeros(length(t),N);
for l=1:N
    for n=1:N
        summa(:,l)=summa(:,l)+
        exp(2*pi*sqrt(-1)*n*l/N-2*pi*sqrt(-1)*(n-N*(n>N/2))/N/delta*t);
    end
end
H=real(1/N*summa);

end

```

Function: dynamical_sys

```
function [f,df]=dynamical_sys(x,sys_arg)
%In this file we give the vector field
[f,df]=Kevrekidis_cartesian(x,sys_arg);
```

Function: Kevrekidis_cartesian

```
function [f,df]=Kevrekidis_cartesian(x,sys_arg)
% kevrekidis example

lambda=sys_arg(1);
epsilon=sys_arg(2);
omega =sys_arg(3);

t1 = x(:,1).^2;
t2 = x(:,2).^2;
t3 = t1+t2;
t4 = t3.^(1./2);
t6 = 1-t4;
t8 = epsilon.*x(:,1);
t9 = 1./t4;
t11 = lambda.*t4.*t6+t8.*t9;
t12 = cos(omega);
t14 = sin(omega);
t16 = x(:,1).*t12-x(:,2).*t14;
t17 = t11.*t16;
t21 = x(:,2).*t12+x(:,1).*t14;
t22 = t11.*t21;
t24 = lambda.*t9;
t31 = 1./t4./t3;
t33 = t24.*t6.*x(:,1)-lambda.*x(:,1)+epsilon.*t9-epsilon.*t1.*t31;
t37 = t11.*t12.*t9;
t38 = t31.*x(:,1);
t44 = t11.*t14.*t9;
t50 = t31.*x(:,2);
t52 = t24.*t6.*x(:,2)-lambda.*x(:,2)-t8.*t50;
f = [t17.*t9,t22.*t9];
df = [t33.*t16.*t9+t37-t17.*t38,t33.*t21.*t9+t44-t22.*t38,...
      t52.*t16.*t9-t44-t17.*t50,t52.*t21.*t9+t37-t22.*t50];
```

Function: Kevrekidis_polar

```
function [f,df] = map(x,sys_arg)

lambda=sys_arg(1);
epsilon=sys_arg(2);
omega=sys_arg(3);
t=sys_arg(4:end)';

f=[lambda*x(:,1).*(1-x(:,1))+epsilon*cos(x(:,2)+t),mod(x(:,2)+omega,2*pi)];
df=[lambda-2*lambda*x(:,1),zeros(length(x(:,1)),1),-epsilon*sin(x(:,2)+t),...
    ones(length(x(:,1)),1)];
```

Function :phani_free

```

function [map ,dmap]=phani_free(x,sys_arg);
m=0.21;
w=1.9;
[rows,cols]=size(x);

options=odeset('RelTol',1e-8,'AbsTol',1e-8);

for i=1:rows
    IC=[x(i,:),x4,reshape(eye(4,4),1,16)];
    [t,y] = ode45('phani_free_eq',[0 1],IC,options,m,A,w);
    map(i,:) = y(end,1:cols);
    temp = reshape(y(end,cols+2:cols+1+(cols+1)^2),cols+1,cols+1);
    dmap(i,:) = reshape(temp(1:cols,1:cols),1,cols^2);
end

```

Function :phani_free_eq

```

function varargout = phani_free_eq(t,y,flag,m,A,w)

switch flag
case ''
    varargout{1} = f(t,y,m,A,w);
case 'events'
    [varargout{1:3}] = events(t,y,m);
otherwise
    error(['Unknown flag ' flag '']);
end

% -----

function dydt = f(t,y,m,A,w)
A=0;
jac = [m-y(3) -1 -y(1) 0; 1 m 0 -A*sin(y(4)); 2*y(1)*y(3) 2*y(2) -1+y(1)^2 0; 0 0 0 0];

dydt = [m*y(1)-y(2)-y(1)*y(3);m*y(2)+y(1)+A*cos(y(4));-y(3)+y(2)^2+y(1)^2*y(3);w;...
        reshape(jac*reshape(y(5:20),4,4),16,1)];

% -----

function [value,isterminal,direction] = events(t,y,m)
value = [];
isterminal = [];
direction = [];

```

A.2 Sweeping Method

```

tin=[2*pi/N:2*pi/N:2*pi];
n=0;
M=0;
index=0;
figure(1)
hold on;

sys_arg=[];
method=1;

```

```

for delta_x4=0.05:0.05:2*pi
    index=index+1;
    sys_arg=delta_x4;
    [xref,tangents,tauref]=guess(x,dim,tin,sys_arg,n,N,interpolation_type);
    x=xref;
    tau=tauref;
    count =0;
    F=1;
    while(max(abs(F))>1e-12 & count<15)
        [F,dF]=Newton(x,dim,tau,N,xref,tin,tangents,sys_arg,interpolation_type);
        temp=[];
        for i =1:dim
            temp=[temp;x(:,i)];
        end
        temp=[temp;tau']-inv(dF)*F;
        for i=1:dim
            x(1:N,i)=temp((i-1)*N+1:i*N);
        end
        tau=temp(dim*N+1:(dim+1)*N)';
        max(abs(F))
        count=count+1;
    end

    M=N;
    [x tout]=resample_update(x,dim,tin,N,M,method,interpolation_type);

    %if we introduce new points in resampling than tin = tout(in case
    %of splines only as the delta will vary and we cannot use fourior)

    [xref,tangents,tauref]=guess(x,dim,tin,sys_arg,0,M,interpolation_type);
    x=xref;
    tau=tauref;
    F=1.0;
    count=0;
    while(max(abs(F))>1e-12 & count<15)
        [F,dF]=Newton(x,dim,tau,N,xref,tin,tangents,sys_arg,interpolation_type);
        temp=[];
        for i =1:dim
            temp=[temp;x(:,i)];
        end
        temp=[temp;tau']-inv(dF)*F;
        for i=1:dim
            x(1:N,i)=temp((i-1)*N+1:i*N);
        end
        tau=temp(dim*N+1:(dim+1)*N)';
        max(abs(F))
        count=count+1;
    end
    X=[x,ones(length(x),1).*mod(delta_x4+19000,2*pi)];
    savefile=['pdata_',num2str(index)]
    save(['z:\fun_work\Danko\Code\New_furior\2D_3D_invariant\Data\sweep_method\phani\',savefile],'X');
    drawnow
    plot3(X(:,1),X(:,2),X(:,3),' ');
    %
    pause
end

```

Function: phani_forced

```
function [map ,dmap]=phani_forced(x,sys_arg);
```

```

m=0.21;
w=1.9;
A=1.0;
x4=sys_arg(1);
[rows,cols]=size(x);

options=odeset('RelTol',1e-8,'AbsTol',1e-8);

for i=1:rows
    IC=[x(i,:),x4,reshape(eye(4,4),1,16)];
    [t,y] = ode45('phani_forced_eq',[0 2*pi/w],IC,options,m,A,w);
    map(i,:) = y(end,1:cols);
    temp = reshape(y(end,cols+2:cols+1+(cols+1)^2),cols+1,cols+1);
    dmap(i,:) = reshape(temp(1:cols,1:cols),1,cols^2);
end

```

Function: phani_forced_eq

```

function varargout = phani_forced_eq(t,y,flag,m,A,w)

switch flag
case ''
    varargout{1} = f(t,y,m,A,w);
case 'events'
    [varargout{1:3}] = events(t,y,m);
otherwise
    error(['Unknown flag ' flag ' ']);
end

% -----

function dydt = f(t,y,m,A,w)

jac = [m-y(3) -1 -y(1) 0; 1 m 0 -A*sin(y(4));...
       2*y(1)*y(3) 2*y(2) -1+y(1)^2 0; 0 0 0 0];

dydt = [m*y(1)-y(2)-y(1)*y(3);m*y(2)+y(1)+A*cos(y(4));...
       -y(3)+y(2)^2+y(1)^2*y(3);w;...
       reshape(jac*reshape(y(5:20),4,4),16,1)];

% -----

function [value,isterminal,direction] = events(t,y,m)
value = [];
isterminal = [];
direction = [];

```

Function :Vander_pol_forced

```

function [f,df]=Vander_pol_forced(x,sys_arg)

F=1;
nu=0.01;
omega = 1.01;
epsilon=0.01;
Time_period_sys = 2*pi/omega;

options=odeset('RelTol',1e-6,'AbsTol',1e-6,'events','off');

```

```

[rows cols]=size(x);
t0=sys_arg;
for i=1:rows
    IC=[x(i,:),t0,reshape(eye(3,3),1,9)];
    [t,y] = ode45('Vander_Pol_eq',[t0 t0+Time_period_sys],IC,options,sys_arg);
    f(i,:) = y(end,1:cols);
    temp = reshape(y(end,cols+2:cols+1+(cols+1)^2),cols+1,cols+1);
    df(i,:) = reshape(temp(1:cols,1:cols),1,cols^2);
end

```

Function :Vander_pol_forced_eq

```

function varargout = Vander_pol_forced_eq(t,y,flag,sys_par)

switch flag
case ''
    varargout{1} = f(t,y);
case 'events'
    [varargout{1:3}] = events(t,y,surface_position);
otherwise
    error(['Unknown flag '' flag ''.']);
end

%Forced VanderPol oscillator

function dy = f(t,y)

F=1;
nu=0.01;
omega = 1.01;
epsilon=0.01;
Time_period_sys = 2*pi/omega;

jac=[0 1 0;...
    -1-2*nu*y(1)*y(2) -nu*(y(1)^2-1) -F*epsilon*omega*sin(omega*y(3));...
    0 0 0];
dy = [y(2);-y(1)-nu*(y(1)^2 -1)*y(2)+epsilon*F*cos(omega*y(3));1;...
    reshape(jac*reshape(y(4:12),3,3),9,1)];

function [value,isterminal,direction] = events(t,y,surface_position)
    value = [y(3)-surface_position(1)];
    direction= [1];
    isterminal =[1];

```

A.3 Two-dimensional Torus

In this section, the matlab codes used to compute the invariant two-dimensional code are given. The flowing list gives the name of the functions used

1. *run* :The main script that calls all other functions.

2. *guess_torus* :Calculates the initial guess *shifts* and the the tangents at every reference point
3. *Newton_bi_torus* :Calculates $\mathbf{F}$ and it Jacobian.
4. *processing_spline* :This function calls all the functions that calculates the Cubic splines and does various operations on it.
 - 4a. *spline_periodic_u* :Calculates the slope and the derivative matrix at each sample point.
 - 4b. *interpolate_spline* :Calculates the interpolated value at any given point on the spline.
 - 4c. *interpolate_spline_deriv_sample* :Calculates the derivative matrix at any given point on the spline.
 - 4d. *interpolate_spline_deriv_time* :Calculates the slope at any given point on the spline.
 - 4e. *interpolate_spline_2nd_deriv_time* :Calculates the second derivative at any given point on the spline.
5. *run_nice* :The main script that calls all other functions for the case in which one can the shift in one of the parameter directions is known.
6. *guess_torus_nice* :Calculates the initial guess *shifts* and the the tangents at every reference point for the case in which one can the shift in one of the parameter directions is known.
7. *Newton_bi_nice* :Calculates $\mathbf{F}$ and it Jacobian for the case in which one can the shift in one of the parameter directions is known.

Script :run

```

dim=3;
Nt=21;
Ns=21;

t_in=[2*pi/Nt:2*pi/Nt:2*pi];
s_in=[2*pi/Ns:2*pi/Ns:2*pi];

% sys_arg=[0.4,5.3];
sys_arg=[];
[xref,tangents_t,tangents_s,sigmaref,tauref]=guess_torus(x,dim,t_in,s_in,sys_arg,0);
x=xref;
tau=tauref;
sigma=sigmaref;

count =0;
F=1;
while(max(abs(F))>1e-12 & count<15)

```

```

[F,dF]=Newton_bi_torus(x,dim,tau,sigma,xref,t_in,s_in,tangents_t,tangents_s,sys_arg);
xn=[reshape(x(:,1),Nt*Ns,1);reshape(x(:,2),Nt*Ns,1);reshape(x(:,3),Nt*Ns,1);...
    reshape(sigma,Nt*Ns,1);reshape(tau,Nt*Ns,1)];
xn2=xn-inv(dF)*F;
x=zeros(Ns,Nt,dim);
sigma=zeros(Ns,Nt);
tau=zeros(Ns,Nt);
for i= 1:dim
    x(:,i)=reshape(xn2((i-1)*Nt*Ns+1:i*Nt*Ns),Ns,Nt);
end
sigma=reshape(xn2(dim*Nt*Ns+1:dim*Nt*Ns+Ns*Nt),Ns,Nt);
tau=reshape(xn2((dim+1)*Nt*Ns+1:end),Ns,Nt);
max(F)
%pause
count=count+1;
end

[xref,tangents_t,tangents_s,sigmaref,tauref]=guess_torus(x,dim,t_in,s_in,sys_arg,0);
x=xref;
tau=tauref;
sigma=sigmaref;

%for resampling
for i= 1:21
    [x_new(:,i,:) tout]=resample_update(X(:,i,:),dim,s_in,N,M,method,interpolation_type);
end

```

Function :guess_torus

```

function [xref,tangents_t,tangents_s,sigma,tau]=guess_torus(x,dim,t_in,s_in,
sys_arg,n)
%Applying the map
temp=x;
for i=1:n
    temp=dynamical_system(temp,dim,length(s_in),length(t_in),sys_arg);
end

xref=temp;% [[x(t,s)],[y(t,s)],[z(t,s)]] N*M points in x,y,z

%Finding the guess values for Tau and Sigma

m_t(length(s_in),length(t_in),dim)=0;
m_s(length(s_in),length(t_in),dim)=0;
m_ts(length(s_in),length(t_in),dim)=0;
tangents_t(length(s_in),length(t_in),dim)=0;
tangents_s(length(s_in),length(t_in),dim)=0;

for j=1:dim
    for i=1:length(s_in)
        temp=processing_spline('spline_periodic',t_in,x(i,:,j));
        m_t(i,:,j)=temp';
    end
    for i=1:length(t_in)
        temp=processing_spline('spline_periodic',s_in,m_t(:,i,j)');
        m_ts(:,i,j)=temp;
        temp=processing_spline('spline_periodic',s_in,x(:,i,j)');
        m_s(:,i,j)=temp;
    end
end

```

```

end
%Input to NEW spline_periodic(row,row)
% return a row of slopes

tangents_t=m_t;
tangents_s=m_s;

%Interpolation

tout=[0:0.01:2*pi];
sout=[0:0.01:2*pi];

xtemp_out=zeros(length(sout),length(t_in),dim);
m_t_out=xtemp_out;
xout =zeros(length(sout),length(tout),dim);

for j=1:dim
    for i=1:length(t_in)
        temp =processing_spline('interpolate',s_in,x(:,i,j)',m_s(:,i,j)',sout);
        xtemp_out(:,i,j)=temp';
        temp=processing_spline('interpolate',s_in,m_t(:,i,j)',m_ts(:,i,j)',sout);
        m_t_out(:,i,j)=temp';
    end
    for i=1:length(sout)
        xout(i,:,j)=processing_spline('interpolate',t_in,xtemp_out(i,:,j),
            m_t_out(i,:,j),tout);
    end
end

y = dynamical_system(x,dim,length(s_in),length(t_in),sys_arg);
%y = map_3D_Langford(x,dim,length(s_in),length(t_in),sys_arg);
tau=zeros(length(s_in),length(t_in));
sigma=zeros(length(s_in),length(t_in));

for j=1:length(t_in)
    for i=1:length(s_in)
        temp=zeros(length(sout),length(tout),dim);
        unity=ones(length(sout),length(tout),dim);
        for k=1:dim
            temp(:,k)=unity(:,k)*y(i,j,k);
        end
        temp=temp-xout;
        temp2=sqrt(temp(:,1).^2+temp(:,2).^2+temp(:,3).^2);
        temp=reshape(temp2,length(tout)*length(sout),1);
        [val,minj]=min(temp);
        col=floor(minj/length(sout))+1;
        row=mod(minj,length(sout));
        if(row==0)
            row=length(sout);
            col=col-1;
        end
        sigma(i,j)=mod(sout(row)-s_in(i),2*pi);
        tau(i,j)=mod(tout(col)-t_in(j),2*pi);
    end
end
end

```

Function :Newton_bi_torus

```
function [F,DF]=Newton_bi_torus(x,dim,tau,sigma,xref,t_in,s_in,tangents_t,
```

```

                                tangents_s,sys_arg)
%M ==> s
%N ==> t
%x(M,N,dim)
%tau(M,N,dim)
%sigma(M,N,dim)

%parameter s and t
check=0;

M=length(s_in);
N=length(t_in);

m_t(M,N,dim)=0;
m_s(M,N,dim)=0;
m_ts(M,N,dim)=0;

dm_t(N,N,M,dim)=0;
dm_s(M,M,N,dim)=0;
dm_ts(M,M,N,dim)=0;

% Using the Old spline code return a vector colum input can be
%either be rows or column
%The modified code return slope as a row vetor takes a two rpw vectors

for j=1:dim
    for i=1:M
        [temp temp_m]=processing_spline('spline_periodic',t_in,x(i,:,j));
        m_t(i,:,j)=temp;
        dm_t(:, :, i, j)=temp_m;
    end

    for i=1:N
        [temp temp_m]=processing_spline('spline_periodic',s_in,m_t(:, i, j));
        m_ts(:, i, j)=temp;
        dm_ts(:, :, i, j)=temp_m; %%dm_ts=d(d(yi)/d(t))d(s))(d(y)/d(t))
        [temp temp_m]=processing_spline('spline_periodic',s_in,x(:, i, j));
        m_s(:, i, j)=temp;
        dm_s(:, :, i, j)=temp_m;
    end
end

%Assuming to check the code
% sigma=rand(M,length(t_in)).*1e-2;
% tau=rand(M,length(t_in)).*1e-2;

%Finding the interpolated value at t+tau,s+sigma

sout=zeros(M,N);
tout=zeros(M,N);
m_t_out=zeros(M,N,dim);
xout =zeros(M,N,dim);
xtemp_out=zeros(M,N,dim);
%This is true because we are not changing the No. of points

for i=1:N
    sout(:,i)=mod(s_in'+sigma(:,i),2*pi);
end
for j=1:M

```

```

    tout(j,:)=mod(t_in+tau(j,:),2*pi);
end

for k=1:dim
    for j=1:N
        for i=1:N
            temp =processing_spline('interpolate',s_in,x(:,i,k)',m_s(:,i,k)',sout(:,j));
            xtemp_out(:,i,k)=temp';
            temp=processing_spline('interpolate',s_in,m_t(:,i,k)',m_ts(:,i,k)',sout(:,j));
            m_t_out(:,i,k)=temp';
        end
        for i=1:M
            xout(i,j,k)=processing_spline('interpolate',t_in,xtemp_out(i,:,k),
                m_t_out(i,:,k),tout(i,j));
        end
    end
end

%Calculation the F vector
[map dmaptemp]=dynamical_system(x,dim,M,N,sys_arg);
delta_x=zeros(M,N,dim);

F1=zeros(M*N*dim,1);
F2=zeros(M*N,1);
F3=zeros(M*N,1);
F2temp=zeros(M,N);
F3temp=zeros(M,N);

for j=1:dim
    temp= map(:,j)-xout(:,j);
    F1((j-1)*N*M+1:j*N*M)=reshape(temp,N*M,1);
    temp=x(:,j)-xref(:,j);
    delta_x(:,j)=temp;
    for i = 1:M
        F2temp(i,:) = F2temp(i,:)+delta_x(i,:,j).*tangents_t(i,:,j);
    end
    for i= 1:N
        F3temp(:,i)=F3temp(:,i)+delta_x(:,i,j).*tangents_s(:,i,j);
    end
end
F2=reshape(F2temp,M*N,1);
F3=reshape(F3temp,M*N,1);

F=[F1;F2;F3];
max(F1)

%Calculating the Jacobian matrix
%dK=>derivative of K w.r.t X(sample points) vector

dtemp=zeros(M,M);
Dy=zeros(M,M*N,N,dim);

for j=1:dim
    %dm_ts=d(d(y)/d(t))d(s))(d(y)/d(t)) w.r t point in the column
    Dm_ts=zeros(M,M*N,N);%derivative w.r.t all points for each column
    %derivative of the column w.r.t all points for each column
    Dm_t=zeros(M,M*N,N);

```

```

for i=1:N
for ro=1:M
col=ro+[0:N-1]*M;
Dm_t(ro,col,i)=dm_t(i,:,ro,j);
end
Dm_ts(:, :, i)=dm_ts(:, :, i, j)*Dm_t(:, :, i);
end

for k=1:N
Dyi_t=zeros(M,M*N,N);
Dyi=zeros(M,M*N,N);
for i=1:N
dtemp(:, :)=processing_spline('interpolate_deriv_sample',s_in,x(:, i, j),
sout(:, k), dm_s(:, :, i, j), '1'); %d(yi)/d(x)
for ro=1:M
Dyi(ro, (i-1)*M+1:i*M, i)=dtemp(ro, :);
end
Dyi_t(:, :, i)=processing_spline('interpolate_deriv_sample',s_in,Dm_t(:, :, i),
sout(:, k), Dm_ts(:, :, i), '2');
end
%Calculate derivative at t=tout(:, k) for each colum
Dy(:, :, k, j)=processing_spline('interpolate_deriv_sample',t_in,Dyi,
tout(:, k), Dyi_t, '3');
end
end

%Calculating derivative w.r.t 'tau' and 'sigma'

dtemp=zeros(M,M);
dy_tau=zeros(M,N);
dy_sigma=zeros(M,N);
mi_s=zeros(M,N);
mi_ts=zeros(M,N);
mi_t=zeros(M,N);
Dy_tau=zeros(M*N,dim);
Dy_sigma=zeros(M*N,dim);
xtemp_out=zeros(M,N);

for j=1:dim
for k=1:N
for i=1:N
temp=processing_spline('interpolate_deriv_time',s_in,x(:, i, j)',
sout(:, k), m_s(:, i, j)');
mi_s(:, i)=temp';
temp=processing_spline('interpolate_deriv_time',s_in,m_t(:, i, j)',
sout(:, k), m_ts(:, i, j)');
mi_ts(:, i)=temp';
temp =processing_spline('interpolate',s_in,x(:, i, j)', m_s(:, i, j)', sout(:, k));
xtemp_out(:, i)=temp';
temp=processing_spline('interpolate',s_in,m_t(:, i, j)', m_ts(:, i, j)', sout(:, k));
mi_t(:, i)=temp';
end
%Calculate derivative at t=tout(:, k) for each colum
for row=1:M
dy_sigma(row, k)=processing_spline('interpolate',t_in,mi_s(row, :),
mi_ts(row, :), tout(row, k));
dy_tau(row, k)=processing_spline('interpolate_deriv_time',t_in,xtemp_out(row, :),
tout(row, k), mi_t(row, :));
end
end

```

```

Dy_tau(:,j)=reshape(dy_tau(:,,:),M*N,1);
Dy_sigma(:,j)=reshape(dy_sigma(:,,:),M*N,1);
end

%-----Checking the derivative
if (check==1)
    sigma2=sigma+rand(M,N).*1e-6;
    %sigma2=sigma;
    %tau2=tau;
    tau2=tau+rand(M,N).*1e-6;

    sout2=zeros(M,N);
    tout2=zeros(M,N);
    xout2 =zeros(M,N);
    xtemp_out2=zeros(M,N);
    m_t_out2=zeros(M,N);
    % This is true because we are not changing the No. of points

    for i=1:N
        sout2(:,i)=mod(s_in'+sigma2(:,i),2*pi);
    end
    for j=1:M
        tout2(j,:)=mod(t_in+tau2(j,:),2*pi);
    end

    for k=1:dim
        for j=1:N
            for i=1:N
                temp =processing_spline('interpolate',s_in,x(:,i,k)',m_s(:,i,k)',
                    sout2(:,j));
                xtemp_out2(:,i)=temp';
                temp=processing_spline('interpolate',s_in,m_t(:,i,k)',m_ts(:,i,k)',
                    sout2(:,j));
                m_t_out2(:,i)=temp';
            end
            for i=1:M
                xout2(i,j,k)=processing_spline('interpolate',t_in,xtemp_out2(i,:),
                    m_t_out2(i,:),tout2(i,j));
            end
        end
    end

    for k=1 :dim
        max(reshape(xout2(:, :,k)-xout(:, :,k),M*N,1)-
            Dy_sigma(:,k).*reshape(sigma2-sigma,M*N,1)-Dy_tau(:,k).*reshape(tau2-tau,M*N,1))
        end
    end
end
%-----%
%Assembling the Jacobian

DF1=[];
DF=zeros(5*M*N,5*M*N);

dmap=zeros(M,M*N,N,dim);

for k=1:dim
    Dy_temp=[];
    for i=1:N

```

```

    Dy_temp=[Dy_temp;Dy(:,:,i,k)];
end
Dy_temp2(:,:,k)=Dy_temp;
end

clear Dy_temp;
DF1=[Dy_temp2(:,:,1),zeros(M*N,M*N),zeros(M*N,M*N);...
     zeros(M*N,M*N),Dy_temp2(:,:,2),zeros(M*N,M*N);...
     zeros(M*N,M*N),zeros(M*N,M*N),Dy_temp2(:,:,3)];

DF1_temp=DF1*0;
for k1=1:dim
    for k2=1:dim
        DF1_temp((k1-1)*M*N+1:k1*M*N,(k2-1)*M*N+1:k2*M*N)=
            diag(reshape(dmaptemp((k1-1)*M+1:k1*M,(k2-1)*N+1:k2*N),1,N*M));
    end
end

DF1=DF1_temp-DF1;

DF1_tau=[];%dim*M*N,M*N
DF1_sigma=[];%dim*M*N,M*N
DF2=[];%M*N,M*N
DF3=[];%M*N,M*N

for i= 1:dim
    DF1_tau=[DF1_tau;diag(Dy_tau(:,i))];
    DF1_sigma=[DF1_sigma;diag(Dy_sigma(:,i))];
    DF2=[DF2,diag(reshape(tangents_t(:,i),1,M*N))];
    DF3=[DF3,diag(reshape(tangents_s(:,i),1,M*N))];
end

%The negative sign is there.F=map-interpolation
DF=[DF1,-DF1_sigma,-DF1_tau;...
    DF2,zeros(M*N,M*N),zeros(M*N,M*N);...
    DF3,zeros(M*N,M*N),zeros(M*N,M*N)];
DF_trial=[DF1,-DF1_sigma,-DF1_tau];

%-----Checking the Jacobian-----%

if(check==1)
    x2=x+rand(M,N,dim)*1e-6;
    sigma2=sigma+rand(M,N).*1e-6;
    tau2=tau+rand(M,N).*1e-6;
    % sigma2=sigma;
    % tau2=tau;
    %x2=x;

    m_t2(M,N,dim)=0;
    m_s2(M,N,dim)=0;
    m_ts2(M,N,dim)=0;

    for j=1:dim
        for i=1:M
            [temp temp_m]=processing_spline('spline_periodic',t_in,x2(i,:,j));
            m_t2(i,:,j)=temp;
        end
    end
end

```

```

for i=1:N
    [temp temp_m]=processing_spline('spline_periodic',s_in,m_t2(:,i,j)');
    m_ts2(:,i,j)=temp';
    [temp temp_m]=processing_spline('spline_periodic',s_in,x2(:,i,j)');
    m_s2(:,i,j)=temp';
end
end

%Finding the interpolated value at t+tau,s+sigma

sout2=zeros(M,N);
tout2=zeros(M,N);
m_t_out=zeros(M,N);
xout2 =zeros(M,N);
xtemp_out=zeros(M,N);
%This is true because we are not changing the No. of points

for i=1:N
    sout2(:,i)=mod(s_in'+sigma2(:,i),2*pi);
end
for j=1:M
    tout2(j,:)=mod(t_in+tau2(j,:),2*pi);
end

for k=1:dim
    for j=1:N
        for i=1:N
            temp =processing_spline('interpolate',s_in,x2(:,i,k)',m_s2(:,i,k)',
                sout2(:,j));
            xtemp_out(:,i)=temp';
            temp=processing_spline('interpolate',s_in,m_t2(:,i,k)',m_ts2(:,i,k)',
                sout2(:,j));
            m_t_out(:,i)=temp';
        end
        for i=1:M
            xout2(i,j,k)=processing_spline('interpolate',t_in,xtemp_out(i,:),m_t_out(i,:),
                tout2(i,j));
        end
    end
end

map2 =dynamical_system(x2,dim,M,N,sys_arg);
delta_x=zeros(M,N,dim);

F1=zeros(M*N*dim,1);
F2=zeros(M*N,1);
F3=zeros(M*N,1);
F2temp=zeros(M,N);
F3temp=zeros(M,N);

for j=1:dim
    temp= map2(:, :,j)-xout2(:, :,j);
    F1((j-1)*N*M+1:j*N*M)=reshape(temp,N*M,1);
    temp=x2(:, :,j)-xref(:, :,j);
    delta_x(:, :,j)=temp;
    for i = 1:M
        F2temp(i, :) = F2temp(i, :)+delta_x(i, :,j).*tangents_t(i, :,j);
    end
end

```

```

    for i= 1:N
        F3temp(:,i)=F3temp(:,i)+delta_x(:,i,j).*tangents_s(:,i,j);
    end
end
F2=reshape(F2temp,M*N,1);
F3=reshape(F3temp,M*N,1);

F2=[F1;F2;F3];

delta_F=F2-F;
temp=[];
temp=x2-x;
delta_x=[];
for i=1:dim
    delta_x=[delta_x;reshape(temp(:,i),M*N,1)];
end

delta_x=[delta_x;reshape((sigma2-sigma),M*N,1);reshape((tau2-tau),M*N,1)];
max(delta_F-(DF*delta_x))
end

```

Function :processing_spline

```

function varargout=processing_spline(flag,tin,x,varargin)

% sample point refer to the points
% time refers to the t parameter
% m slope w.r.t t
% m_x slopr w.r.t sample point data

switch flag
case 'spline_periodic'
    [m,m_x]=spline_periodic_u(tin,x);
    varargout{1}= m;
    varargout{2}=m_x;
case 'interpolate'
    Tout=varargin{2};
    m=varargin{1};
    varargout{1} = interpolate_spline(tin,x,m,Tout);
case 'interpolate_deriv_sample'
    Tout=varargin{1};
    dm=varargin{2};
    flag=varargin{3};
    varargout{1}=interpolate_spline_deriv_sample(tin,x,dm,Tout,flag);
case 'interpolate_deriv_time'
    Tout=varargin{1};
    m=varargin{2};
    varargout{1}=interpolate_spline_deriv_time(tin,x,m,Tout);
case 'interpolate_2nd_deriv_time'
    Tout=varargin{1};
    m=varargin{2};
    varargout{1}=interpolate_spline_2nd_deriv_time(tin,x,m,Tout);
end

```

Function :spline_periodic_u

```

function [m,dm] =spline_periodic_u(t,x)

```

```

%return a row vector and the jacobian
%takes two row vectors

N=length(t);
dt=diff([0,t,t(1)+t(end)]);
dx=diff([x(N),x,x(1)]);
rhs(1:N)=6*(dx(2:N+1)./dt(2:N+1).^2+dx(1:N)./dt(1:N).^2);
coeff=sparse(1:N,1:N,4./dt(1:N)+4./dt(2:N+1),N,N)+...
    sparse([N,1:N-1],1:N,2./dt(1:N),N,N)+...
    sparse(1:N,[N,1:N-1],2./dt(2:N+1),N,N);
drhs=sparse(1:N,1:N,6./dt(1:N).^2-6./dt(2:N+1).^2,N,N)+...
    sparse([N,1:N-1],1:N,6./dt(1:N).^2,N,N)+...
    sparse(1:N,[N,1:N-1],-6./dt(2:N+1).^2,N,N);

m=(rhs/coeff);
dm=(drhs/coeff);

```

Function :interpolate_spline_deriv_sample

```

function dxout=interpolate_spline_deriv_sample(tin,xin,dm,tout,flag)

%Each row of xout is the derivative w.r.t all other sample points
%flag=1 vector
%flag=2 2D matrix
%flag=3 3D matrix

N=length(tin);
s=[0 tin];

switch flag
case '1'
    y_d=sparse([1:N+1],[N,1:N],1,N+1,N);
    dxout=zeros(length(tout),N);
    dy_d=[dm(end,:);dm];%making it cyclic

    for i=1:length(tout)
        k=max(find(s<=mod(tout(i),2*pi)));
        t = (mod(tout(i),2*pi)-s(k))/(s(k+1)-s(k));
        p1 = (t-1)^2*(1+2*t); q1 = t*(t-1)^2;
        p2 = t^2*(3-2*t); q2 = t^2*(t-1);
        dxout(i,:)=y_d(k,:)*p1+y_d(k+1,:)*p2+(s(k+1)-s(k))*(dy_d(k,:)*q1+
            dy_d(k+1,:)*q2);
    end

case '2'
    y_d=[xin(end,:);xin];%making it cyclic
    [i j]=size(xin);
    dxout=zeros(length(tout),j);
    dy_d=[dm(end,:);dm];%making it cyclic

    for i=1:length(tout)
        k=max(find(s<=mod(tout(i),2*pi)));
        t = (mod(tout(i),2*pi)-s(k))/(s(k+1)-s(k));
        p1 = (t-1)^2*(1+2*t); q1 = t*(t-1)^2;
        p2 = t^2*(3-2*t); q2 = t^2*(t-1);
        dxout(i,:)=y_d(k,:)*p1+y_d(k+1,:)*p2+(s(k+1)-s(k))*(dy_d(k,:)*q1+
            dy_d(k+1,:)*q2);
    end
end

```

```

case '3'
    [i j k]=size(xin);
    y_d(:,:,2:k+1)=xin;%making it cyclic
    y_d(:,:,1)=xin(:,:,end);

    dy_d(:,:,2:k+1)=dm;%making it cyclic
    dy_d(:,:,1)=dm(:,:,end);
    dxout=zeros(length(tout),j);

    for c=1:length(tout)
        kc=max(find(s<=mod(tout(c),2*pi)));
        t = (mod(tout(c),2*pi)-s(kc))/(s(kc+1)-s(kc));
        p1 = (t-1)^2*(1+2*t); q1 = t*(t-1)^2;
        p2 = t^2*(3-2*t); q2 = t^2*(t-1);
        dxout(c,:)=y_d(c,:,kc).*p1+y_d(c,:,kc+1).*p2+(s(k+1)-s(k)).*[dy_d(c,:,kc).*q1+
            dy_d(c,:,kc+1).*q2];
    end
end

```

Function :interpolate_spline_deriv_time

```

function dxout=interpolate_spline_deriv_time(tin,xin,m,tout)

s=[0 tin];
y=[xin(end) xin];
dy=[m(end) m];
dxout=zeros(1,length(tout));
for i=1:length(tout)
    k=max(find(s<=mod(tout(i),2*pi)));
    t = (mod(tout(i),2*pi)-s(k))/(s(k+1)-s(k));
    p1t = 6*t*(t-1); q1t = (3*t-1)*(t-1);
    p2t = -6*t*(t-1); q2t = t*(3*t-2);
    dxout(i)=(y(k)*p1t+y(k+1)*p2t)/(s(k+1)-s(k))+dy(k)*q1t+dy(k+1)*q2t;
end

```

Function :interpolate_spline_2nd_deriv_time

```

function dxout=interpolate_spline_2nd_deriv_time(tin,xin,m,tout)

s=[0 tin];
y=[xin(end) xin];
dy=[m(end) m];
dxout=zeros(1,length(tout));
for i=1:length(tout)
    k=max(find(s<=mod(tout(i),2*pi)));
    t = (mod(tout(i),2*pi)-s(k))/(s(k+1)-s(k));
    p1tt = 6*(2*t-1); q1tt = 6*t-4;
    p2tt = -6*(2*t-1); q2tt = (6*t-2);
    dxout(i)=(y(k)*p1tt+y(k+1)*p2tt)/(s(k+1)-s(k))^2+(dy(k)*q1tt+
        dy(k+1)*q2tt)/(s(k+1)-s(k));
end

```

Function :eg_danko2

```

function [map,dmap]=eg_danko2(x,sys_arg)

[M N dim]=size(x);
%M row
%N col
map=zeros(M,N,dim);
dmap=zeros(3*M,3*N);

options=odeset('RelTol',1e-6,'AbsTol',1e-6,'events','off');

for i=1:M
    for j=1:N
        IC=[x(i,j,1),x(i,j,2),x(i,j,3),reshape(eye(3,3),1,9)];
        [t,y]=ode45('eg_danko2_eq',[0 1],IC,options);
        map(i,j,1:3)=y(end,1:3);
        dmap(i,j)=y(end,4);
        dmap(i,N+j)=y(end,7);
        dmap(i,2*N+j)=y(end,10);
        dmap(M+i,j)=y(end,5);
        dmap(M+i,N+j)=y(end,8);
        dmap(M+i,2*N+j)=y(end,11);
        dmap(2*M+i,j)=y(end,6);
        dmap(2*M+i,N+j)=y(end,9);
        dmap(2*M+i,2*N+j)=y(end,12);
    end
end

```

Function :eg_danko2_eq

```

function varargout = eg_danko(t,y,flag)

angle=2*pi/22;
switch flag
    case ''
        varargout{1} = f(t,y);
    case 'events'
        [varargout{1:3}]=events(t,y,angle);
    otherwise
        error(['Unknown flag ' flag '.']);
end

% -----

function dydt = f(t,y)
y1=y(1);
y2=y(2);
y3=y(3);

a=5;
omega=5.3;
alpha=1.0;
beta=0.4;
x1=sqrt(y1^2+y2^2)-a;
c=alpha*y1^2/(a+x1)^2+beta;

jac=[y3/(y1^2+y2^2)^(1/2)-y3*y1^2/(y1^2+y2^2)^(3/2),...
    -y3*y1/(y1^2+y2^2)^(3/2)*y2-omega,...

```

```

y1/(y1^2+y2^2)^(1/2);...
-y3*y1/(y1^2+y2^2)^(3/2)*y2*omega,...
y3/(y1^2+y2^2)^(1/2)-y3*y2^2/(y1^2+y2^2)^(3/2),...
y2/(y1^2+y2^2)^(1/2);...
(2*alpha*y1/(y1^2+y2^2)-2*alpha*y1^3/(y1^2+y2^2)^2)*y3*
(1-((y1^2+y2^2)^(1/2)-a)^2)-
2*(alpha*y1^2/(y1^2+y2^2)+beta)*y3*((y1^2+y2^2)^(1/2)-a)*
y1/(y1^2+y2^2)^(1/2)-y1/(y1^2+y2^2)^(1/2),...
-2*alpha*y1^2/(y1^2+y2^2)^2*y2*y3*(1-((y1^2+y2^2)^(1/2)-a)^2)-
2*(alpha*y1^2/(y1^2+y2^2)+
beta)*y3*((y1^2+y2^2)^(1/2)-a)*y2/(y1^2+y2^2)^(1/2)-y2/(y1^2+y2^2)^(1/2),...
(alpha*y1^2/(y1^2+y2^2)+beta)*(1-((y1^2+y2^2)^(1/2)-a)^2)];

dydt =[y3*y1/(a+x1)-y2*omega;...
y3*y2/(a+x1)+y1*omega;...
c*y3*(1-x1^2)-x1;...
reshape(jac*reshape(y(4:12),3,3),9,1)];

function [value,isterminal,direction]=events(t,y,angle)
value = atan2(y(2),y(1))-angle;
if (t>1e-5)
    isterminal=1;
else
    isterminal=0;
end
direction=1;

```

Script :run_nice

```

dim=3;
Nt=20;
Ns=30;
index=1;

t_in=[2*pi/Nt:2*pi/Nt:2*pi];
s_in=[2*pi/Ns:2*pi/Ns:2*pi];

sys_arg(1)=0.05;
[xref,tangents_t,sigmaref,tauref]=guess_torus_nice(x,dim,t_in,s_in,sys_arg,0);

for A=0.05:0.05:1.06
    index=index+1;
    sys_arg(1)=A
    x=xref;
    tau=tauref;
    sigma=sigmaref;
    count =0;
    F=1;
    while(max(abs(F))>1e-7 & count<15)
        [F,dF]=Newton_bi_nice(x,dim,tau,sigma,xref,t_in,s_in,tangents_t,sys_arg);
        xn=[reshape(x(:,:),1),Nt*Ns,1];reshape(x(:,:),2),Nt*Ns,1);
        reshape(x(:,:),3),Nt*Ns,1);...
        reshape(tau,Nt*Ns,1)];
        xn2=xn-dF\F;
        x=zeros(Ns,Nt,dim+1);
        tau=zeros(Ns,Nt);
        for i= 1:dim
            x(:,:,i)=reshape(xn2((i-1)*Nt*Ns+1:i*Nt*Ns),Ns,Nt);
        end
    end
end

```

```

x(:, :, dim+1)=xref(:, :, dim+1);
tau=reshape(xn2(dim*Nt*Ns+1:end),Ns,Nt);
max(abs(F))
count=count+1;
end

savefile=['phani_data_', num2str(index)]
save(['z:\fun_work\Danko\Code\2_Torus_3_space\bicubic\Data
      \phani_forced\ ', savefile], 'x', 'tau', 'sigma');

xref=x;
tauref=tau;
sigmaref=sigma;
end

```

Function :guess_torus_nice

```

function [xref,tangents_t,sigma,tau]=guess_torus_nice(x,dim,t_in,s_in,
                                                    sys_arg,n)
%This is the nice case i.e we know the shifts in the y3 direction for the
%forced Vanderpol oscillator
%we know the shift in s i.e sigma
%The dimension is 2

omega=1.9;
%Applying the map
temp=x;
for i=1:n
    temp=dynamical_system(temp,dim,length(s_in),length(t_in),sys_arg);
end

xref=temp;% [[x(t,s)],[y(t,s)],[z(t,s)]] N*M points in x,y,z

%Finding the guess values for Tau
m_t(length(s_in),length(t_in),dim)=0;
m_ts(length(s_in),length(t_in),dim)=0;
tangents_t(length(s_in),length(t_in),dim)=0;

for j=1:dim
    for i=1:length(s_in)
        temp=processing_spline('spline_periodic',t_in,x(i,:,j));
        m_t(i,:,j)=temp';
    end
    for i=1:length(t_in)
        temp=processing_spline('spline_periodic',s_in,m_t(:,i,j)');
        m_ts(:,i,j)=temp;
        temp=processing_spline('spline_periodic',s_in,x(:,i,j)');
        m_s(:,i,j)=temp;
    end
end
%Input to NEW spline_periodic(row,row)
% return a row of slopes

tangents_t=m_t;

%Interpolation

tout=[0:0.01:2*pi];

```

```

sout=mod(s_in+omega,2*pi);%omega is the shift in sigma(known)

xtemp_out=zeros(length(sout),length(t_in),dim);
m_t_out=xtemp_out;
xout =zeros(length(sout),length(tout),dim);

for j=1:dim
    for i=1:length(t_in)
        temp =processing_spline('interpolate',s_in,x(:,i,j)',
            m_s(:,i,j)',sout);
        xtemp_out(:,i,j)=temp';
        temp=processing_spline('interpolate',s_in,m_t(:,i,j)',
            m_ts(:,i,j)',sout);
        m_t_out(:,i,j)=temp';
    end
    for i=1:length(sout)
        xout(i,:,j)=processing_spline('interpolate',t_in,xtemp_out(i,:,j),
            m_t_out(i,:,j),tout);
    end
end

y = dynamical_system(x,dim,length(s_in),length(t_in),sys_arg);
tau=zeros(length(s_in),length(t_in));
sigma=ones(length(s_in),length(t_in)).*mod(omega,2*pi);

for j=1:length(t_in)
    for i=1:length(s_in)
        temp=zeros(length(sout),length(tout),dim);
        unity=ones(length(sout),length(tout),dim);
        for k=1:dim
            temp(:,:,k)=unity(:,:,k)*y(i,j,k);
        end
        temp=temp-xout;
        temp2=sqrt(temp(:,:,1).^2+temp(:,:,2).^2);
        temp=reshape(temp2,length(tout)*length(sout),1);
        [val,minj]=min(temp);
        col=floor(minj/length(sout))+1;
        row=mod(minj,length(sout));
        if(row==0)
            row=length(sout);
            col=col-1;
        end
        tau(i,j)=mod(tout(col)-t_in(j),2*pi);
    end
end
end

```

Function :Newton_bi_nice

```

%The extra nice case for 2 torus in 3 D space.
%dim=2;

function [F,DF]=Newton_bi_nice(x,dim,tau,sigma,xref,t_in,
    s_in,tangents_t,sys_arg)

%M ==> s
%N ==> t
%x(M,N,dim)
%tau(M,N,dim)
%sigma(M,N,dim)

```

```

%parameter s and t
check=0;

M=length(s_in);
N=length(t_in);

m_t(M,N,dim)=0;
m_s(M,N,dim)=0;
m_ts(M,N,dim)=0;

dm_t(N,N,M,dim)=0;
dm_s(M,M,N,dim)=0;
dm_ts(M,M,N,dim)=0;

for j=1:dim
    for i=1:M
        [temp temp_m]=processing_spline('spline_periodic',
            t_in,x(i,:,j));
        m_t(i,:,j)=temp;
        dm_t(:, :, i,j)=temp_m;
    end

    for i=1:N
        [temp temp_m]=processing_spline('spline_periodic',s_in,m_t(:,i,j));
        m_ts(:,i,j)=temp;
        dm_ts(:, :, i,j)=temp_m; %%dm_ts=d(d(d(yi)/d(t))d(s))(d(y)/d(t))
        [temp temp_m]=processing_spline('spline_periodic',s_in,x(:,i,j));
        m_s(:,i,j)=temp;
        dm_s(:, :, i,j)=temp_m;
    end
end
%Finding the interpolated value at t+tau,s+sigma
%sigma is known

sout=zeros(M,N);
tout=zeros(M,N);
m_t_out=zeros(M,N,dim);
xout =zeros(M,N,dim);
xtemp_out=zeros(M,N,dim);
%This is true because we are not changing the No. of points

for i=1:N
    sout(:,i)=mod(s_in'+sigma(:,i),2*pi);
end
for j=1:M
    tout(j,:)=mod(t_in+tau(j,:),2*pi);
end

for k=1:dim
    for j=1:N
        for i=1:N
            temp =processing_spline('interpolate',s_in,x(:,i,k)',m_s(:,i,k)',
                sout(:,j));
            xtemp_out(:,i,k)=temp;
            temp=processing_spline('interpolate',s_in,m_t(:,i,k)',m_ts(:,i,k)',
                sout(:,j));
            m_t_out(:,i,k)=temp;
        end
    end
end

```

```

end
for i=1:M
    xout(i,j,k)=processing_spline('interpolate',t_in,xtemp_out(i,:,k),
        m_t_out(i,:,k),tout(i,j));
end
end
end

%Calculation the F vector
[map dmaptemp]=dynamical_system(x,dim,M,N,sys_arg);
delta_x=zeros(M,N,dim);

F1=zeros(M*N*dim,1);
F2=zeros(M*N,1);
F2temp=zeros(M,N);

for j=1:dim
    temp= map(:,j)-xout(:,j);
    F1((j-1)*N*M+1:j*N*M)=reshape(temp,N*M,1);
    temp=x(:,j)-xref(:,j);
    delta_x(:,j)=temp;
    for i = 1:M
        F2temp(i,:) = F2temp(i,:)+delta_x(i,:,j).*tangents_t(i,:,j);
    end
end
F2=reshape(F2temp,M*N,1);

F=[F1;F2];
%max(F1)

%Calculating the Jacobian matrix
%dK=>derivative of K w.r.t X(sample points) vector

dtemp=zeros(M,M);
Dy=zeros(M,M*N,N,dim);

for j=1:dim
    %dm_ts=d(d(d(y)/d(t))d(s))(d(y)/d(t)) w.r t point in the column
    Dm_ts=zeros(M,M*N,N);%derivative w.r.t all points for each column
    %derivative of the column w.r.t all points for each column
    Dm_t=zeros(M,M*N,N);
    for i=1:N
        for ro=1:M
            col=ro+[0:N-1]*M;
            Dm_t(ro,col,i)=dm_t(i,:,ro,j);
        end
        Dm_ts(:,i)=dm_ts(:,i,j)*Dm_t(:,i,i);
    end

    for k=1:N
        Dy_i_t=zeros(M,M*N,N);
        Dy_i=zeros(M,M*N,N);
        for i=1:N
            dtemp(:,:)=processing_spline('interpolate_deriv_sample',s_in,x(:,i,j),
                sout(:,k),dm_s(:,i,j),'1');%d(yi)/d(x)
            for ro=1:M
                Dy_i(ro,(i-1)*M+1:i*M,i)=dtemp(ro,:);
            end
        end
    end
end

```

```

        Dy_i_t(:, :, i) = processing_spline('interpolate_deriv_sample', s_in,
            Dm_t(:, :, i), sout(:, k), Dm_ts(:, :, i), '2');
    end
    %Calculate derivative at t=tout(:,k) for each colum
    Dy(:, :, k, j) = processing_spline('interpolate_deriv_sample', t_in,
        Dy_i, tout(:, k), Dy_i_t, '3');
    end
end

%Calculating derivative w.r.t 'tau'

dtemp = zeros(M, M);
dy_tau = zeros(M, N);
mi_s = zeros(M, N);
mi_ts = zeros(M, N);
mi_t = zeros(M, N);
Dy_tau = zeros(M * N, dim);
xtemp_out = zeros(M, N);

for j = 1:dim
    for k = 1:N
        for i = 1:N
            temp = processing_spline('interpolate', s_in, x(:, i, j)',
                m_s(:, i, j)', sout(:, k));
            xtemp_out(:, i) = temp;
            temp = processing_spline('interpolate', s_in, m_t(:, i, j)',
                m_ts(:, i, j)', sout(:, k));
            mi_t(:, i) = temp;
        end
        %Calculate derivative at t=tout(:,k) for each colum
        for row = 1:M
            dy_tau(row, k) = processing_spline('interpolate_deriv_time', t_in,
                xtemp_out(row, :), tout(row, k), mi_t(row, :));
        end
    end
    Dy_tau(:, j) = reshape(dy_tau(:, :), M * N, 1);
end

%----Checking the derivative
if (check == 1)
    %tau2=tau;
    tau2 = tau + rand(M, N) .* 1e-6;

    sout2 = zeros(M, N);
    tout2 = zeros(M, N);
    xout2 = zeros(M, N);
    xtemp_out2 = zeros(M, N);
    m_t_out2 = zeros(M, N);
    % This is true because we are not changing the No. of points

    for i = 1:N
        sout2(:, i) = mod(s_in' + sigma(:, i), 2 * pi);
    end
    for j = 1:M
        tout2(j, :) = mod(t_in + tau2(j, :), 2 * pi);
    end

    for k = 1:dim
        for j = 1:N

```

```

for i=1:N
    temp =processing_spline('interpolate',s_in,x(:,i,k)',
        m_s(:,i,k)',sout2(:,j));
    xtemp_out2(:,i)=temp';
    temp=processing_spline('interpolate',s_in,m_t(:,i,k)',
        m_ts(:,i,k)',sout2(:,j));
    m_t_out2(:,i)=temp';
end
for i=1:M
    xout2(i,j,k)=processing_spline('interpolate',t_in,xtemp_out2(i,:),
        m_t_out2(i,:),tout2(i,j));
end
end
end

for k=1 :dim
    max(reshape(xout2(:, :, k)-xout(:, :, k),M*N,1)-
        Dy_tau(:,k).*reshape(tau2-tau,M*N,1))
        %pause
    end
end
%-----%
%Assembling the jacobian

DF=zeros((dim+1)*M*N,(dim+1)*M*N);
DF1=zeros(dim*M*N,dim*M*N);

dmap=zeros(M,M*N,N,dim);

%derivatives of the interpolated surface
for k=1:dim
    Dy_temp=[];
    for i=1:N
        Dy_temp=[Dy_temp;Dy(:, :, i,k)];
    end
    Dy_temp2(:, :, k)=Dy_temp;
end

clear Dy_temp;
for k= 1:dim
    DF1((k-1)*M*N+1:k*M*N,(k-1)*M*N+1:k*M*N)=Dy_temp2(:, :, k);
end
% DF1=[Dy_temp2(:, :, 1),zeros(M*N,M*N);...
%      zeros(M*N,M*N),Dy_temp2(:, :, 2)];

%derivatives of the map
DF1_temp=DF1*0;
for k1=1:dim
    for k2=1:dim
        DF1_temp((k1-1)*M*N+1:k1*M*N,(k2-1)*M*N+1:k2*M*N)=
            diag(reshape(dmaptemp((k1-1)*M+1:k1*M,(k2-1)*N+1:k2*N),1,N*M));
    end
end

DF1=DF1_temp-DF1;
DF1_tau=[];%dim*M*N,M*N
DF2=[];%M*N,M*N

```

```

for i= 1:dim
    DF1_tau=[DF1_tau;diag(Dy_tau(:,i))];
    DF2=[DF2,diag(reshape(tangents_t(:, :, i),1,M*N))];
end

%The negative sign is there.F=map-interpolation
DF=[DF1,-DF1_tau;...
    DF2,zeros(M*N,M*N)];

DF_trial=[DF1,-DF1_tau];

%-----Checking the Jacobian-----%

if (check==1)
    x2=x(:, :, 1:2)+rand(M,N,dim).*1e-6;
    x2(:, :, 3)=x(:, :, 3);
    tau2=tau+rand(M,N).*1e-6;
    sigma2=sigma;
    % tau2=tau;
    %x2=x;

    m_t2(M,N,dim)=0;
    m_s2(M,N,dim)=0;
    m_ts2(M,N,dim)=0;

    for j=1:dim
        for i=1:M
            [temp temp_m]=processing_spline('spline_periodic',t_in,
                x2(i, :, j));
            m_t2(i, :, j)=temp;
        end
        for i=1:N
            [temp temp_m]=processing_spline('spline_periodic',s_in,m_t2(:, i, j));
            m_ts2(:, i, j)=temp;
            [temp temp_m]=processing_spline('spline_periodic',s_in,x2(:, i, j));
            m_s2(:, i, j)=temp;
        end
    end

    %Finding the interpolated value at t+tau,s+sigma

    sout2=zeros(M,N);
    tout2=zeros(M,N);
    m_t_out=zeros(M,N);
    xout2 =zeros(M,N);
    xtemp_out=zeros(M,N);
    %This is true because we are not changing the No. of points

    for i=1:N
        sout2(:,i)=mod(s_in'+sigma2(:,i),2*pi);
    end
    for j=1:M
        tout2(j,:)=mod(t_in+tau2(j,:),2*pi);
    end

    for k=1:dim
        for j=1:N
            for i=1:N

```

```

        temp =processing_spline('interpolate',s_in,x2(:,i,k)',m_s2(:,i,k)',
                                ,sout2(:,j));
        xtemp_out(:,i)=temp';
        temp=processing_spline('interpolate',s_in,m_t2(:,i,k)',m_ts2(:,i,k)',
                                ,sout2(:,j));
        m_t_out(:,i)=temp';
    end
    for i=1:M
        xout2(i,j,k)=processing_spline('interpolate',t_in,xtemp_out(i,:),
                                        m_t_out(i,:),tout2(i,j));
    end
end
end

map2 =dynamical_system(x2,dim,M,N,sys_arg);
delta_x=zeros(M,N,dim);

F1=zeros(M*N*dim,1);
F2=zeros(M*N,1);
F2temp=zeros(M,N);

for j=1:dim
    temp= map2(:, :, j)-xout2(:, :, j);
    F1((j-1)*N*M+1:j*N*M)=reshape(temp,N*M,1);
    temp=x2(:, :, j)-xref(:, :, j);
    delta_x(:, :, j)=temp;
    for i = 1:M
        F2temp(i,:) = F2temp(i,:)+delta_x(i, :, j).*tangents_t(i, :, j);
    end
end
F2=reshape(F2temp,M*N,1);

F2=[F1;F2];
delta_F=F2-F;
temp=[];
temp=x2-x;
delta_x=[];
for i=1:dim
    delta_x=[delta_x;reshape(temp(:, :, i),M*N,1)];
end

delta_x=[delta_x;reshape((tau2-tau),M*N,1)];
max(delta_F-(DF*delta_x))
end

function [map,dmap]=Vander_pol(x,sys_arg)

options=odeset('RelTol',1e-6,'AbsTol',1e-6,'events','off');
[M N dim]=size(x);
for i= 1:M
    for j=1:N
        IC=[x(i,j,1),x(i,j,2),x(i,j,3),reshape(eye(3,3),1,9)];
        [t,y] = ode45('Vander_pol_forced_eq',[0 1],IC,options,sys_arg);
        map(i,j,:) = [y(end,1),y(end,2)];
        dmap(i,j)=y(end,4);
        dmap(i,N+j)=y(end,7);
        dmap(M+i,j)=y(end,5);
        dmap(M+i,N+j)=y(end,8);
    end
end
end

```

Function :Vander_pol_forced_eq

```
function varargout = Vander_pol_forced_eq(t,y,flag,sys_par)

switch flag
case ''
    varargout{1} = f(t,y);
case 'events'
    [varargout{1:3}] = events(t,y);
otherwise
    error(['Unknown flag ' flag '.']);
end

%Forced VanderPol oscillator

function dy = f(t,y)

F=1;
nu=0.01;
omega = 1.01;
epsilon=0.01;

jac=[0 1 0;...
    -1-2*nu*y(1)*y(2) -nu*(y(1)^2-1) -epsilon*F*sin(y(3));...
    0 0 0];

dy = [y(2);-y(1)-nu*(y(1)^2 -1)*y(2)+epsilon*F*cos(y(3));omega;...
    reshape(jac*reshape(y(4:12),3,3),9,1)];

function [value,isterminal,direction] = events(t,y)

value = [mod(y(3),2*pi)-pi];
isterminal = [0];
direction = [1];
```

Function :phani_forced

```
function [map ,dmap]=phani_forced(x,sys_arg);
m=0.21;
w=1.9;
A=sys_arg(1);
[M,N,dim]=size(x);

options=odeset('RelTol',1e-6,'AbsTol',1e-6,'events','off');

for i= 1:M
    for j=1:N
        IC=[x(i,j,1),x(i,j,2),x(i,j,3),x(i,j,4),reshape(eye(4,4),1,16)];
        [t,y] = ode45('phani_forced_eq',[0 1],IC,options,m,A,w);
        map(i,j,:) = [y(end,1:3)];
        dmap(i,j)=y(end,5);
        dmap(i,N+j)=y(end,9);
        dmap(i,2*N+j)=y(end,13);
        dmap(M+i,j)=y(end,6);
        dmap(M+i,N+j)=y(end,10);
        dmap(M+i,2*N+j)=y(end,14);
        dmap(2*M+i,j)=y(end,7);
```

```

    dmap(2*M+i,N+j)=y(end,11);
    dmap(2*M+i,2*N+j)=y(end,15);
end
end

```

Function :phani_forced_eq

```

function varargout = phani_forced_eq(t,y,flag,m,A,w)

switch flag
case ''
    varargout{1} = f(t,y,m,A,w);
case 'events'
    [varargout{1:3}] = events(t,y,m);
otherwise
    error(['Unknown flag ' flag '.']);
end

% -----

function dydt = f(t,y,m,A,w)

jac = [m-y(3) -1 -y(1) 0; 1 m 0 -A*sin(y(4)); 2*y(1)*y(3) 2*y(2) -1+
        y(1)^2 0;0 0 0 0];

dydt = [m*y(1)-y(2)-y(1)*y(3);m*y(2)+y(1)+A*cos(y(4));-y(3)+y(2)^2+
        y(1)^2*y(3);w;...
        reshape(jac*reshape(y(5:20),4,4),16,1)];

% -----

function [value,isterminal,direction] = events(t,y,m)
value = [];
isterminal = [];
direction = [];

```

Vita

Gunjan Singh Thakur was born in December of 1975 in New Delhi, India. He received his B.Tech degree in 2000 from Indian Institute of Technology, Guwahati in Mechanical Engineering. After graduation he worked in Tata Engineering as a Design Engineer. In 2002, he came to the department of Engineering Science and Mechanics at Virginia Polytechnic and State University for his masters. He completed his degree in 2004.