A Numerical Study of the Lorenz and Lorenz-Stenflo Systems

TOMMY EKOLA

Doctoral Thesis
Stockholm, Sweden 2005
Abstract

In 1998 the Swedish mathematician Warwick Tucker used rigorous interval arithmetic and normal form theory to prove the existence of a strange attractor in the Lorenz system. In large parts, that proof consists of computations implemented and performed on a computer. This thesis is an independent numerical verification of the result obtained by Warwick Tucker, as well as a study of a higher-dimensional system of ordinary differential equations introduced by the Swedish physicist Lennart Stenflo.

The same type of mapping data as Warwick Tucker obtained is calculated here via a combination of numerical integration, solving optimisation problems and a coordinate change that brings the system to a normal form around the stationary point in the origin. This data is collected in a graph and the problem of determining the existence of a strange attractor is translated to a few graph theoretical problems. The end result, after the numerical study, is a support for the conclusion that the attractor set of the Lorenz system is a strange attractor and also for the conclusion that the Lorenz-Stenflo system possesses a strange attractor.

Keywords. Warwick Tucker, Strange attractor, Lorenz equations, Lorenz-Stenflo equations, Lorenz attractor, Lorenz-Stenflo attractor, Dynamical systems, Normal form theory.

2000 Mathematics Subject Classification. Primary 37D45; Secondary 34C20, 65P20.
Sammanfattning

År 1998 bevisade den svenske matematikern Warwick Tucker med hjälp av rigorös intervallaritmetik och normalformsteori att en "strange attractor" existerar för Lorenz-systemet. Till stora delar består detta bevis av beräkningar som programmerats på och utförts av en dator. Denna avhandling är en oberoende numerisk undersökning av Warwick Tuckers resultat och en studie av ett högredimensionellt system av ordinära differentialekvationer framtaget av den svenske fysikern Lennart Stenflo.

Samma typ av avbildningsdata som Warwick Tucker använde i sitt bevis beräknas här via en kombination av numerisk integration, lösande av optimeringsproblem och ett koordinatbyte som omvandlar systemet till ett normalformssystem runt den stationära punkten i origo. Dessa data används i en graf för att avgöra om en "strange attractor" existerar. Slutresultatet, efter den numeriska studien, stöder slutsatsen att en "strange attractor" existerar för Lorenzsystemet och samma slutsats för Lorenz-Stenflo systemet.
Preface

In 1963 the American meteorologist Edward Lorenz presented a simple system of three ordinary differential equations with remarkably intricate solutions. Almost all solutions of this system tend to a complicated wing-sized subset of the phase space and on this subset, also known as the Lorenz attractor, orbits seem to exhibit a complex behaviour described in technical terms as a strange attractor. At the time this was a new phenomena and starting from the 1970’s, when there were more systematic studies performed, the Lorenz system has been one of the most prominent model problems in studying chaotical dynamical systems. It even has it that the famous mathematician Steven Smale mentioned proving the existence of a strange attractor in the Lorenz system as one of the eighteen great problems in mathematics for the twenty-first century [Mathematical Intelligencer 20 (1998), no. 2, pages 7–15].

Although the basic mechanism of the Lorenz system has more or less been well understood since the introduction of the geometric Lorenz model by John Guckenheimer and Robert F. Williams in 1979 the existence of a strange attractor remained an unsolved problem. During the 1980’s and early 1990’s several partial results were proved on the chaotical behaviour of the Lorenz equations but it was only in 1998 that the Swedish mathematician Warwick Tucker presented a proof via a novel combination of rigorous interval arithmetic and normal form theory.

The purpose of this thesis is twofold: First, it is an independent numerical verification of the result obtained in the Lorenz case. It was felt that although Warwick Tucker profess to have a rigorous proof a large part of the calculations in his proof are performed on a computer and there is no easy way to ensure that his computer code is 100% correct. Secondly, the main focus in this thesis will be on an extended system of differential equations, called the Lorenz-Stenflo system. This will demonstrate that the method of Warwick Tucker also works in a higher-dimensional situation.

Needless to say, this thesis relies heavily on the work of Warwick Tucker but also contains modifications and improvements. The main features are:

1. The power series of the coordinate change functions $\phi$ and $\psi$, associated with the normal form system, are calculated with a high-order truncation which changes the remainder term estimates, especially that for $\psi$, and it was also natural to present the derivation in a new form.
2. A more tight analysis of the difference between the normal form system and the linear system is needed to properly estimate cones as they flow through a cube centred at the origin and are flattened at the exit faces of the cube. A side effect of this is that the degree of the non-linearity of the normal form system can be chosen lower than before.

3. The invariant cone field is constructed in a more structured way and this results in narrower cones. This construction also works in a higher dimensional setting.

4. The total expansiveness of the return map is interpreted as a graph problem and solved in a more standard way.

An important difference between this work and the work of Warwick Tucker is the usage of numerical integrator routines instead of rigorous interval arithmetic. This particular decision was made early on and the main motivation was to get a result closer to the true result by a combination of numerically integrating orbits and solving certain optimisation problems. Even if this means that the results obtained are not rigorous they can still be considered reliable as no orbit is integrated over any longer period of time.

* 

I thank my advisor Professor Michael Benedicks for his continuous support during the years. My thanks also extend to the Department of Mathematics, KTH, for providing for me and offering a stimulating environment. Moreover, a big thank-you is in order to family and friends.

Tommy Ekola
Stockholm, 11 March 2005
# Contents

## 1 The two systems
- 1.1 A background to the Lorenz system ........................................ 1
- 1.2 The Lorenz attractor .......................................................... 2
- 1.3 The geometric Lorenz model ............................................... 7
- 1.4 The method of Warwick Tucker ........................................... 10
- 1.5 Non-uniform hyperbolicity .................................................. 15
- 1.6 The Lorenz-Stenflo system .................................................. 16

## 2 The numerical method .......................................................... 19
- 2.1 A short summary .................................................................. 19
- 2.2 Step 0 ................................................................................. 23
- 2.3 Step 1 and 2 ....................................................................... 30
- 2.4 Step 3 and 4 ....................................................................... 32
- 2.5 Step 5 and 6 ....................................................................... 41
- 2.6 Step 7 ................................................................................. 52
- 2.7 Step 8 and 9 ....................................................................... 59
- 2.8 Step 10 and 11 ................................................................... 67
- 2.9 Step 12 and 13 ................................................................... 76
- 2.10 Step 14 and 15 ................................................................. 85

## 3 The coordinate change around the origin .................................. 91
- 3.1 The normal form method ....................................................... 91
- 3.2 The coordinate change function $\phi$ .................................... 92
- 3.3 The coordinate change function $\psi$ .................................... 105
- 3.4 The non-linear function $G$ .................................................. 114
- 3.5 The normal form flow ......................................................... 120
- 3.6 The Lorenz system .............................................................. 130

## 4 Results .................................................................................... 135

## A References ............................................................................ 141
Chapter 1

The two systems

1.1 A background to the Lorenz system

The first step towards a modern theory of meteorology was taken by the Norwegian meteorologist Vilhelm Bjerknes in 1904 [Meteorologische Zeitschrift 21 (1904), pages 1–7] when he wrote down the physical laws governing the state of the atmosphere as a set of integro-differential equations. The idea was that if you measured the relevant physical quantities at one time you could make a forecast of future weather by (numerically) solving the equations as an initial-value problem. Although this set the foundation of meteorology Bjerknes realized that it was impossible to perform the necessary calculations by hand. So his work had no immediate practical importance.

During the First World War the British meteorologist Lewis F. Richardson took up this idea. He simplified the equations and developed extensive numerical procedures to solve the equations [Weather prediction by numerical process, Cambridge University Press (1922)]. Again the practical problems of actually performing the calculations prevented any real test of the method.

Meteorology underwent a rapid development during the Second World War when accurate forecasts were needed and after the war the first computers were used to forecast the weather by numerical computations. One of these initiatives were organized by the famous mathematician John von Neumann at the Institute for Advanced Study and the goal was to develop a hierarchy of numerical models each building on the other with increasing complexity to clarify and capture the essential aspects of the forecast problem. This continued in the 1950’s and there was great optimism as the models were refined, although the results were still mediocre. It was during this time that a controversy broke out among meteorologists between a group who favoured traditional forecasts based on statistical methods as opposed to numerical methods. To show that this statistical group was incorrect and that it was not merely a lack of data behind the flawed predictions, the American meteorologist Edward Lorenz set out to find a simple non-periodic system of equations
that he could use to test the statistical method against and hopefully get a contradictory result. His first successful test problem was to take two horizontal surfaces and the equations of the atmosphere between them. Each atmospheric variable was expanded in an orthogonal series and truncated. After some attempts and simplifications Lorenz arrived at a system of twelve equations which had the desired non-periodic behaviour. The statistical method failed for this problem so the result was as expected.

In his experiments Lorenz wanted to examine one particular solution in greater detail. Instead of starting the computer from the beginning he chose an intermediate value typed out by the computer as a start value. When he returned to the computer after about two simulated weather months the result of the new computer run completely disagreed with the earlier run. He soon realized that the cause of this was that the typed out values had only three decimals while the computer used six decimals internally. This small difference in initial-values would be amplified each iteration and consequently yield a totally different solution. Lorenz concluded that if the weather system exhibited the same phenomena long-term forecasts would be impossible. Still not satisfied, Lorenz wanted to find an even simpler system with this kind of sensitive behaviour but his attempts to simplify the system were unsuccessful. When he later visited Barry Saltzman (Yale University) and discussed his problem he was shown a system of seven ordinary differential equations that Saltzman had derived from a similar physical problem [Journal for the Atmospheric Sciences 19 (1962), pages 329–341] and which had a solution that didn’t settle down. Since four of the components approached zero, Lorenz eliminated these and arrived at a system of three equations

\[
\begin{align*}
\dot{x} &= -\sigma x + \sigma y, \\
\dot{y} &= \varrho x - y - xz, \\
\dot{z} &= xy - \beta z,
\end{align*}
\]

with parameter values \( \sigma = 10, \varrho = 28 \) and \( \beta = 8/3 \). This system was studied by Lorenz in his seminal paper [Journal for the Atmospheric Sciences 20 (1963), pages 130–141] and it turned out to have a surprisingly intricate behaviour.


1.2 The Lorenz attractor

A first step in understanding the Lorenz system, or any system of ordinary differential equations, is to study its stationary points (points where the right-hand side
of (1) vanishes). Between the stationary points the flow is regular and after classifying the local behaviour of orbits near the stationary points it is often possible to extrapolate a global picture of the system by filling in the blanks.

The Lorenz system has three stationary points

\[(0,0,0)\text{ and } C^{\pm} = (\pm \sqrt{\beta(\varrho - 1)}, \pm \sqrt{\beta(\varrho - 1)}, \varrho - 1).\]

When we linearise the system at the origin \((0,0,0)\) we get a linear system \(\dot{x} = Ax\), where the matrix \(A\) has the following eigenvalues

\[\lambda_1 \approx 11.83, \quad \lambda_2 \approx -22.83 \quad \text{and} \quad \lambda_3 \approx -2.67.\]

This means that the origin is a saddle point with two contracting directions and one expanding direction. At the points \(C^{\pm}\) the linearised system has a matrix with eigenvalues

\[\lambda_1 \approx -13.85 \quad \text{and} \quad \lambda_{2,3} \approx 0.09 \pm 10.19i.\]

This is a saddle point with one contracting direction and two directions where the orbits are spiralling out.

By drawing the stationary points and their local orbits in a combined picture we get Figure 1.1. It is not clear from this figure how the orbits fit together and form a complete phase portrait.
Orbits of the system are also confined to be inside a sphere $x^2 + y^2 + (z - (\sigma + \varrho))^2 = C^2$ for large enough $C$. (This can easily be proved by showing that the vector field of the Lorenz system points inwards on the sphere.) This means that orbits must stay within a bounded region of the phase space and since all stationary points are saddle points a typical orbit cannot reach a steady-state solution but must instead, perhaps, approach a limit cycle or even something more complicated.

To gain a better understanding of how orbits behave we pick some rather arbitrary starting point $(x_0, y_0, z_0)$ and study its orbit. We choose the starting point close to the origin. The orbit approaches the critical point $C^+$ and makes an arc around it (Figure 1.2a). It is then very rapidly thrown over to a region close to the critical point $C^-$ where it slowly spirals out (Figure 1.2b). Eventually the orbit moves far enough from $C^-$ to cross over to the $C^+$-side. The orbit now makes its second turn around $C^+$ (Figure 1.2c). Since the orbit isn’t close to $C^+$ it will rapidly move over to $C^-$ again, where it makes two laps (Figure 1.2d). The pattern then repeats. The orbit comes over to the $C^+$-side, this time closer to $C^+$ and therefore it makes a few turns before returning to the $C^-$-spiral (Figure 1.2e).

This type of behaviour, alternating between the two spirals with no apparent pattern, is observed for almost all orbits and they all seem to approach the same limiting figure regardless of starting point (the exceptions seem to be orbits that

Figure 1.2: The evolution of an orbit that starts in the point $(1, 1, 1)$. 
1.2. THE LORENZ ATTRACTOR

Figure 1.3: This is a simplistic figure illustrating the attractor set as a surface. The two wing-sized sheets around the two stationary points appear to merge somewhere between the stationary points. The grey arrows illustrate how orbits move on the attractor. (happen to land on a stationary point).

In order to form a picture of how this attractor set looks like, Edward Lorenz suggested that one imagines the set to be a surface where two sheets merge as the two spirals join together on the way down towards the origin (see Figure 1.3). The two sheets cannot of course merge due to the uniqueness of orbits (only one orbit can pass through a non-stationary point) and they just appear to do so but are in reality two sheets very close to each other. As this pair of sheets return they again blend together as four sheets, then eight sheets and so on. Ultimately, the attractor set must consist of a complex of infinitely many sheets and the way they come together implies that the cross-section of the attractor has a Cantor set-like structure with uncountably many sheets. A truly complicated limiting behaviour for a system, which is so analytically simple.

Apart from the circular motion, orbits also tend to spiral outwards as they move around anyone of the two stationary points until thrown over to the other spiral. The factor that seems to determine the changeover from one spiral to the other is how far out on the spiral an orbit is. To measure this, Lorenz used the maximum $z$-value of an orbit as it revolves around the spiral one turn. By then studying a typical orbit for a long time he obtained a sequence $\{z_i\}$ of such relative maximal $z$-values and plotted $z_{i+1}$ against $z_i$ to find out how the position of an orbit on a spiral determines its position the next turn. This resulted in the diagram found in Figure 1.4(a). What can be concluded from this diagram is that there indeed appears to be an almost perfect relation between successive maximal $z$-values as the points line up to an essentially one-dimensional curve. The 2-to-1 correspondence of this map is caused by the fact that an orbit with a certain maximal $z$-value can stem from either one of the spirals.

Interestingly, this one-dimensional map between maximal $z$-values is also everywhere expanding (has a slope strictly greater than 1 everywhere) and therefore has
Figure 1.4: The diagram (a) plots the relative maximum $z$-value of an orbit that makes a loop around one of the stationary points versus the same quantity of the previous loop. This diagram can be compared to the diagram (b) of a stepwise linear expanding map.

a similar dynamics as the so-called tent map in Figure 1.4(b),

$$f(z) = \begin{cases} 2z, & \text{if } z \leq \frac{1}{2}, \\ 2 - 2z, & \text{if } z \geq \frac{1}{2}. \end{cases}$$

If we have two points with slightly different $z$-values $z$ and $z + \varepsilon$ then after $n$ iterations with $f$ they have separated by $2^n \varepsilon$. The same type of behaviour can then be translated to orbits on the attractor and this helps to explain the phenomena Lorenz observed of sensitivity on initial conditions.

We have a somewhat paradoxical situation: Almost all orbits will tend to the same limiting set but on that set they have a divergent behaviour. This new phenomena was also later observed by other scientists, e.g. S. Smale and R. V. Plykin, in their work and in 1971 the mathematical physicist David Ruelle and his coauthor Floris Takens [Communications in Mathematical Physics 20 (1971), pages 167–192] introduced the loosely defined notion of a “strange attractor” to denote an attractor set with properties resembling those of the Lorenz attractor. Although there is no authoritative definition of the term strange attractor the following definition must be considered uncontroversial. (See the article by John Milnor [Communications in Mathematical Physics 99 (1985), pages 177–195] for a discussion of the general concept of an attractor.)

**Definition 1.** A set $\Lambda$ is called a strange attractor if

1. The set $\Lambda$ is compact.
2. There exists an open set $U$ that contains $\Lambda$ and is attracted to $\Lambda$, i.e. the distance between the orbit of $x$ and the set $\Lambda$ tends to 0 as the time $t \to \infty$ for all $x$ in $U$.

3. There is a dense orbit in $\Lambda$, i.e. there is a point in $\Lambda$ whose orbit visits every neighbourhood of every point in $\Lambda$.

4. For every point in $\Lambda$ and any neighbourhood of that point there exists a point in the neighbourhood such that the two points eventually separate to a fixed distance $\delta > 0$ under the flow.

Over the years a number of attempts have been made to rigorously prove that the Lorenz attractor possesses chaotical behaviour. In this vein we find the work of S.P. Hastings and W.C. Troy [Bulletin of the American Mathematical Society 27, no. 2, pages 298–303] that shows the existence of a homoclinic orbit and the work of K. Mischaikow and M. Mrozek [Bulletin of the American Mathematical Society 32 (1995), no. 1, pages 66–72] that demonstrates the existence of horse shoes. Both these properties imply a chaotical behaviour of the system in a small subset of the phase space. However, the question of whether the Lorenz attractor is a strange attractor or not was only resolved in 1998 by Warwick Tucker in his thesis [The Lorenz attractor exists, Uppsala universitet] and subsequent article [Foundations of Computational Mathematics 2 (2002), pages 53–117].

1.3 The geometric Lorenz model

In 1979 John Guckenheimer and Robert F. Williams [Publ. Math. IHES 50 (1979), pages 59–72] introduced a model of the Lorenz system that was aimed at qualitatively understanding the system and help to explain the presence of an attractor set. (It is noteworthy that at about the same time three Russian mathematicians V.S. Afraimovich, V.V. Bykov and L.P. Sil’nikov [Dokl. Acad. Sci. USSR 234 (1977), pages 336–339] made a similar analysis.)

In their model the origin plays an important role as the only stationary point that orbits on the attractor flow arbitrary close to. This is a fact that can be inferred from studying orbits very closely. The local behaviour of orbits near the origin resembles the linearised picture we illustrated in Figure 1.1. There is a two-dimensional manifold $W^s(0)$ consisting of points with orbits that converge to the origin and also a one-dimensional unstable manifold $W^u(0)$ of points with orbits that originate from the origin if they are followed backwards. These two manifolds correspond to the stable plane and unstable line of the linearised system.

On the stable manifold the sizes of the two negative eigenvalues $\lambda_2 < \lambda_3 < 0$ of the linearised system indicate how orbits will approach the origin. The orbits have a stronger approach rate in the direction that corresponds to $\lambda_2$ and this causes them to converge towards the origin in the $\lambda_3$-direction, as seen in Figure 1.5. Although the orbits that actually belong to the stable manifold form a negligible subset of all orbits they do provide important information since any orbit that flows close to the
CHAPTER 1. THE TWO SYSTEMS

Figure 1.5: The figure in (a) illustrates the stable manifold $W^s(0)$ and unstable manifold $W^u(0)$ of the origin, and also shows how orbits on the stable manifold tend to the origin almost exclusively from one direction. Compare this with the figure in (b) of the linearised system.

Much of the coming analysis of the Lorenz system involves the origin and this makes it convenient, at this time, to introduce a linear change of coordinates, as suggested by Warwick Tucker,

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ A_+ & A_- & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix},$$

where $A_{\pm} = (\sigma + 1 \pm \sqrt{\sigma + 1}^2 + 4\sigma(\sigma - 1)) / 2\sigma$, that makes the new coordinate axes coincide with the stable and unstable directions of the origin. In these new coordinates the linear part of the Lorenz system is in a diagonal form and the whole system is written as

$$\begin{cases} \dot{x}_1 = \lambda_1 x_1 - k_1 (x_1 + x_2)x_3, \\ \dot{x}_2 = \lambda_2 x_2 + k_1 (x_1 + x_2)x_3, \\ \dot{x}_3 = \lambda_3 x_3 + (x_1 + x_2)(k_2 x_1 + k_3 x_2), \end{cases}$$

where $\lambda_1 \approx 11.83$, $\lambda_2 \approx -22.83$, $\lambda_3 \approx -2.67$, $k_1 \approx 0.29$, $k_2 \approx 2.18$ and $k_3 \approx -1.28$. In the future we will only deal with this form of the Lorenz system.

The geometric Lorenz model starts by picking a so-called Poincaré plane. This plane, or surface, is chosen to be transversal to the vector field of the system at
1.3. THE GEOMETRIC LORENZ MODEL

Figure 1.6: A schematic figure illustrating how a certain rectangle in the Poincaré plane evolves in the geometric Lorenz model and intersects the Poincaré plane again as two disjoint sets inside the rectangle. The stable manifold of the origin intersects the rectangle along a curve where this mapping is discontinuous.

all points near the attractor (no orbit intersects the plane tangentially) and in the Lorenz case the plane \( x_3 = \rho - 1 \) is a suitable choice as it contains the centre points \( C^\pm \) of the two spirals. In this plane there is a rectangle that is mapped back to the Poincaré plane according to Figure 1.6. The rectangle is cut in half by the stable manifold of the origin and the two halves return as two thin elongated cusp-shaped sets inside the rectangle. The mapping from the Poincaré plane and back is called the first-return map (or the Poincaré map) and the effect it has on either of the two halves is that it expands the half-rectangle in one direction and compresses it in the perpendicular direction. Near the intersection of the rectangle and the stable manifold points flow close to the origin and this deforms the rectangle to the cusp shape it has when returning.

The model assumes that the rectangle admits a stable foliation according to the following definition.

**Definition 2.** A collection of regular and disjoint curves \( \{ \gamma_\alpha \} \) is called a stable foliation of the rectangle under the Poincaré map \( R \) if

1. the union of all curves is the whole rectangle (except, perhaps, for a set of Lebesgue measure zero),

2. the curves are mapped with the Poincaré map to each other, i.e. \( R(\gamma_\alpha) \subset \gamma_\beta \)

3. the Poincaré map restricted to a single curve is a strict contraction.

An individual curve in the collection is called a stable leaf.

With a stable foliation the dynamics can essentially be reduced to study one point on each stable leaf since all points on the same leaf tend to each other under iteration and therefore have the same dynamics.
For simplicity John Guckenheimer and Robert F. Williams assumed that the rectangle is $0 \leq x_1, x_2 \leq 1$ and that the Poincaré map can be written as

$$R(x_1, x_2) = (f(x_1), g(x_1, x_2)).$$

In this case the stable leaves are all lines with constant $x_1$-coordinate and the dynamics can be restricted to the line $x_2 = 0$ and the function $f$. If the function $f$ has the following properties

- $f$ is locally eventually onto (i.e., for every non-empty open set $U$ there is a positive integer $n$ such that $f^n(U) = [0, 1]$),
- $f$ has a single discontinuity at $x_1 = c$ and is strictly increasing on $[0, c)$ and $(c, 1)$,
- $f(c^-) = 1$ and $f(c^+) = 0$ where $f(0) < c < f(1)$,
- $f'(x) \to \infty$ as $x \to c^\pm$,

then they showed that the limiting set in the rectangle is what we call a strange attractor and, hence, that the whole Lorenz attractor is a strange attractor. It should be remarked that “locally eventually onto” implies property 4 in the definition of strange attractor and that Guckenheimer and Williams showed that if $f'(x) > \sqrt{2}$ for all $x$ then this follows.

### 1.4 The method of Warwick Tucker

The main stumbling block to realizing the geometric Lorenz model is to rigorously handle the stationary point in the origin and orbits flowing close to it. As was mentioned in the last section there is no upper bound on the time an orbit can
spend in a neighbourhood of the origin and this makes it impossible to use any
standard numerical ODE solver to investigate this class of orbits.

To introduce the novel approach by Warwick Tucker to this problem it is instruc-
tive to go back to the thesis of the famous French mathematician Henri Poincaré
(reprinted in [Œuvres de Henri Poincaré, TOME I, Paris: Gauthier-Villars, 1928])
where he studied systems of ordinary differential equations around a stationary
point. His idea was to introduce a formal analytic change of coordinates
\( z = \psi(x) \) that transforms the system to its linearised system. In the Lorenz case this would
mean that the Lorenz system

\[
\frac{d}{dt} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + F(x),
\]

where \( F(x) \) denotes the second degree terms, is transformed into the linear system

\[
\frac{d}{dt} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}.
\]

This presents an interesting way of dealing with the problem of the origin: When
an orbit enters the region where the coordinate change is defined, the method is
to transform to the linear system and use it to flow through the region (the linear
system can be solved explicitly) and then when exiting the region transform back
to the original system and continue.

The problem with this method of attack is that the formal change of coordinates
only exists if there is no so-called resonance between the linear coefficients \( \lambda_1, \lambda_2 \)
and \( \lambda_3 \), i.e. if \( \lambda_i \neq n_1\lambda_1 + n_2\lambda_2 + n_3\lambda_3 \) for non-negative integers \( n_1, n_2 \) and \( n_3 \)
where \( n_1 + n_2 + n_3 \geq 2 \). Although this could possibly be true for the parameter
values used for the Lorenz system the infinite number of conditions involved in
non-resonance make the set of “bad” parameters dense in the parameter space and
the existence not robust to small parameter changes. Moreover, non-resonance only
proves that a formal change of coordinates exists and the question of convergence
and radius of convergence leads to a deeper study of near resonance (see e.g. Chap-
ter 5 in [Geometrical Methods in the Theory of Ordinary Differential Equations,
Springer Verlag (1983)]).

Warwick Tucker overcame these difficulties in his method by using a variant of
the coordinate change above, that is not a full linearisation of the Lorenz system
but instead transforms the system to the form

\[
\frac{d}{dt} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} + G(z),
\]

where the non-linear term \( G(z) \) satisfies \( |G_i(z)| \leq C|z_1|^{10}|z_2| + |z_3|^{10} \) near the
origin. This special form of the non-linear term is chosen so that \( G(z) \) is particularly
small close to the stable manifold, which in these coordinates is \( z_1 = 0 \), and the
unstable manifold \( z_2 = z_3 = 0 \), in a region where orbits that flow close to the
origin must spend most of its time. A consequence is that orbits of the normal
form system (4) can be approximated well by the corresponding orbits of the linear
system (3) even though there are orbits with unbounded integration time. The new
transformation does not completely eliminate the problem of resonance but makes
it only necessary to check a finite set of integer linear combinations and Tucker was
able to show that the analytic coordinate change is defined in the cube \( \{|x_i| \leq \frac{1}{10}\} \)
centred around the origin.

In Chapter 3 there is a detailed derivation of estimates involved in this kind of
transformation for the Lorenz-Stenflo system (see next section) but the calculations
are almost identical in the Lorenz case. See also the original article by Warwick

With the analysis of the behaviour close to the origin in place Warwick Tucker
uses a method of invariant rectangles and cones in the Poincaré plane to prove the
existence of a strange attractor. The assumptions involved in the arguments of the
proof are the following:

1. An invariant collection of rectangles \( \{B_i\} \) in the Poincaré plane that cover
the attractor. Each rectangle in this collection is mapped with the Poincaré
map \( R \) inside the collection.

2. On all rectangles there is an invariant regular cone field \( C(x) \). The invariance
means that the cone \( C(x) \) is mapped with the differential of the Poincaré
map \( dR_x \) inside the cone at the image point \( R(x) \), i.e. \( dR_x(C(x)) \subset C(R(x)) \)
for all \( x \) excluding the points on the stable manifold.

3. Tangent vectors inside the invariant cones are eventually expanded, i.e. the
inequality \( \|dR^n_x(v)\| \geq C\|v\|r^n \) holds for almost all \( x \), all tangent vectors \( v \)
in \( C(x) \) and where \( r > 1 \) and \( C > 0 \).

4. On the rectangles that intersect the stable manifold of the origin, and therefore
contain the discontinuity of the Poincaré map, the tangent vectors must be
expanded by at least a factor of 2 before returning to these rectangles.

The existence of an invariant regular cone field together with the fact that some
iterate of the Poincaré map is area contracting (this follows from that the divergence
of the Lorenz flow has a negative constant value and that the flow is transverse to the
Poincaré plane) show that there exists a stable foliation inside the rectangle covering
(see Section 3 in [Invariant Manifolds, Springer Verlag (1977)]) as predicted by the
geometric model. Property 3 and 4 imply that the induced map \( f \) has a positive
Lyapunov exponent and is eventually onto. This shows that \( f \) possesses a strange
attractor and therefore that the Poincaré map attractor and the whole Lorenz
attractor are strange attractors.

The collection of rectangles in the Poincaré plane is generated dynamically by
selecting rectangles from a grid of equal-sized rectangles covering the whole plane.
1.4. THE METHOD OF WARWICK TUCKER

The procedure starts with picking one rectangle that supposedly cover a small part of the attractor set in the Poincaré plane and let that rectangle be the initial collection. Then all rectangles that cover the image of the start rectangle are added to the collection and for each new rectangle the same procedure is repeated. This goes on until all rectangles in the collection have been processed at which time the collection is invariant.

A rectangle is mapped from the Poincaré plane and back by dividing its orbit into a number of intermediate planes and flowing it from one plane to the next by an Euler step. In each intermediate plane a rectangle, or a set of rectangles, is created that contains the image of the rectangle from the previous plane. In this way the rectangle, or rectangles, returning to the Poincaré plane is guaranteed to contain the image of the original rectangle. The main parts of this mapping scheme are illustrated in Figure 1.8.

If a rectangle contains the stable manifold of the origin it will flow down and hit the cube \( \{ |x_i| \leq \frac{1}{10} \} \) centred around the origin. The part of the rectangle that lands inside the top face of the cube flows through the cube via the normal form.
transformation as illustrated in Figure 1.9.

On the start rectangle, as the collection of rectangles is initialised, there is also a cone with angle aperture $[0^\circ, 10^\circ]$ relative to the $x_1$-direction. As the rectangle flows between two consecutive planes the equations of variation

$$\frac{d}{dt} D\phi(x, t) = Df(\phi(x, t)) D\phi(x, t)$$

is also estimated and in the next plane a cone is created that contains all images of the cone under the stepwise differential map.

When the cone returns to the Poincaré plane it contains all images of the start cone and on the rectangles hit by the cone this return cone is picked as start cones. The new cones then go through the same thing. If a cone returns outside the cone on the hit rectangle then the cone on the hit rectangle is enlarged to contain the image cone and the new cone needs to be flown again. Eventually this results in an
invariant piecewise constant cone field and the existence of a regular invariant cone field. The last conclusion is easier to grasp if the single cone on each rectangle in the covering is replaced with two cones, an inner and outer cone, with the following properties:

- The outer cone on a rectangle contains the inner cone on that rectangle as well as the inner cones on neighbouring rectangles.
- The outer cone of a rectangle is mapped with the differential map inside the inner cones of the rectangles the image of the outer cone hits.

A regular invariant cone field can then be fitted between the two cones and seamlessly cross the boundary between two rectangles.

In each transition between two intermediate planes there is also an estimate of the smallest expansion factor a vector inside the cone is subjected to. These expansion factors are multiplied together to form an expansion estimate of the whole mapping from the Poincaré plane and back.

All mapping data are then analysed and properties 3 and 4 are verified.

### 1.5 Non-uniform hyperbolicity

The Lorenz system is probably the most well-known example in the science literature of a system with chaotical behaviour and the work of Warwick Tucker is the first method to rigorously prove the existence of a strange attractor for this system and with these parameter values. It should, however, be remarked that the Lorenz system is not a prototype for all kinds of chaotical behaviour and that the method outlined in the previous section is not applicable to all systems.

An illuminating example is a system constructed by the German scientist Otto Rössler [Physics Letters A 57 (1976), no. 5, pages 397–398]

\[
\begin{align*}
\dot{x} &= -y - z, \\
\dot{y} &= x + ay, \\
\dot{z} &= b + z(x - c).
\end{align*}
\]

With parameter values \(a = b = 0.2\) and \(c = 5.7\) this system exhibits what seems to be a strange attractor, but the underlying mechanism of this attractor is different from the Lorenz attractor. In the Lorenz case the attractor set can be seen as a surface that is sliced up by the stable manifold of the origin and the two sheets are merged after a revolution around two stationary points (see Figure 1.3). A similar model can be made of the Rössler attractor but now the surface is not cut in two pieces but instead folded and thereafter the two sheets are merged.

A study of the relative maximal \(x\)-values \(\{x_i\}\) of a typical orbit of the Rössler system is shown in Figure 1.10(a) where \(x_{i+1}\) is plotted against \(x_i\). Notice that the one-dimensional curve in this case is of a different type than the one obtained for the Lorenz system. The folding introduces a quadratic-like maximum and dynamically
it is more accurate to compare the induced one-dimensional map with a family of quadratic maps

\[ f(x) = 1 - ax^2 \]

than a piecewise expanding map. The dynamics of the family of quadratic maps is considerably more difficult to analyse and important studies were performed by M. Jakobson [Communications in Mathematical Physics 81 (1981), no. 1, pages 39–88] and by M. Benedicks and L. Carleson [Annals of Mathematics 122 (1985), no. 1, pages 1–25]. A recent overview can be found in an article by M. Lyubich [Notices of the AMS 47 (2000), no. 9, pages 1042–1052].

Figure 1.10: In diagram (a) the relative maximum \( x \)-value \( x_{i+1} \) of a typical orbit of the Rössler system is plotted against \( x_i \). This almost uniquely defined mapping of one maximal \( x \)-value to the next is of the same type as the quadratic map \( f(x) = 1 - ax^2 \) found in diagram (b).

Even for the Lorenz system there are parameters where such a quadratic behaviour can be observed. One such parameter triple is \( \sigma = 10 \), \( \varrho = 55 \) and \( \beta = \frac{8}{3} \).

1.6 The Lorenz-Stenflo system

This thesis concentrates on a generalisation of the Lorenz system to four dimensions that was derived by the Swedish physicist Lennart Stenflo. In two articles [Physica Scripta 43 (1991), pages 599–600] and [Physica Scripta 53 (1996), pages 83–84]
1.6. THE LORENZ-STENFLO SYSTEM

Figure 1.11: This figure compares the $z_{\text{max}}$-diagram of the Lorenz-Stenflo system with $\sigma = 10$, $\varrho = 30$, $\beta = 8/3$ and $s = 30$ (black curve) with the Lorenz system where $s = 0$ (grey curve).

Lennart Stenflo studied the equations that govern atmospheric waves. By using a low-frequency and short-wavelength approximation he managed to derive a set of simplified equations and this system was reduced even further by removing one of the independent spatial variables and using a high latitude approximation (the earth rotation vector is constant). He then used the same strategy as Edward Lorenz and substituted a truncated orthogonal series solution into the equations. After normalisation of the variables the following system of ordinary differential equations arose

\[
\begin{align*}
\dot{x} &= \sigma(y - x) + sv, \\
\dot{y} &= r x - xz - y, \\
\dot{z} &= xy - \beta z, \\
\dot{v} &= -x - \sigma v,
\end{align*}
\]

where $v = v(t)$ is a new fourth variable and $s$ is a parameter.

The Lorenz system and this new Lorenz-Stenflo system share the same basic set-up. The Lorenz-Stenflo system has three stationary points $(0,0,0,0)$ and

\[
C^{\pm} = \left(\pm \sqrt{\beta z_s/(1 + s/\sigma^2)}, \pm \sqrt{\beta z_s(1 + s/\sigma^2)}, z_s, \mp \sqrt{\beta z_s/(s + \sigma^2)} \right)
\]

where $z_s = r - 1 - s/\sigma^2$. For a suitable range of parameters the origin and $C^\pm$ are saddle points where the origin has a one-dimensional unstable manifold and a three-dimensional stable manifold while both $C^\pm$ have a one-dimensional stable manifold and a three-dimensional unstable manifold where orbits have an outwards spiralling motion.

Almost all orbits of the system behave in a similar way and tend to an attractor set. This set consist of orbits that circle around the two points $C^\pm$ and occasionally flow close to the origin; the same picture observed in the Lorenz case. After studying
the attractor set more closely it seems plausible to expect that the Lorenz-Stenflo system has a strange attractor and this fact becomes even more so by considering the diagram in Figure 1.11 that plots the same type of $z_{\text{max}}$-diagram as in Figure 1.4. Take note of the similar shape of the one-dimensional curve for the parameter values $\sigma = 10$, $\varrho = 30$, $\beta = 8/3$ and $s = 30$ compared to when $s = 0$ (the Lorenz system). These particular parameter values and the diagram in Figure 1.11 are found in the article [Physica Scripta 55 (1997), pages 394–402] and offer a suitable generalisation of the Lorenz system with the classical parameter values.

In this thesis we redo the work Warwick Tucker did for the Lorenz system in the case of the Lorenz-Stenflo system. To this end some aspects of his work need to be generalised to cope with the higher-dimensional case and although our approach is to a large extent the same as his there are some differences. The most controversial part is probably that the rigorous interval arithmetic is replaced with a shooting algorithm. This issue together with more comments on other aspects are further discussed in the preface. A quick summary of the coming chapters is that Chapter 2 describes the programs and the numerical methods in detail. The most important parts of the programs are also listed with comments. Chapter 3 examines the analytical change of coordinates around the origin. This differ only in some details from Warwick Tucker’s analysis. The whole outcome of the numerical study is summarised in Chapter 4.
Chapter 2

The numerical method

This chapter describes the numerical method that we use when we study the two dynamical systems. In the first section we give a short summary of the various steps involved and thus try to give an overview of the whole numerical study. Then, we go into detail and study the individual programs. In chapter 4 we present the results obtained for the Lorenz and Lorenz-Stenflo systems when the computer programs are run.

2.1 A short summary

The whole numerical method is divided into 15 consecutive steps (however, some of the steps are grouped in pairs and must be run in parallel). Results from one step, or a pair of steps, are passed to the next step by means of files to make it easy to check intermediate results. The details of this are displayed in a diagram in Figure 2.5 after the summarised presentation of the steps.

Throughout the text we use the Lorenz-Stenflo system as the object of study since this higher dimensional system brings out certain multidimensional aspects more clearly than the Lorenz system does.

Step 0

Step 0 is a collection of different Maple programs that output basic Matlab routines. These routines are all connected with the return map or the coordinate change functions around the origin. It is necessary to generate these routines before any of the real computational steps begin.

Step 1 and 2

First the Poincaré plane (which in the Lorenz-Stenflo case is a three-dimensional hyperplane) is subdivided into a regular grid of non-overlapping same-sized boxes.
The algorithm starts by picking a point in the Poincaré plane (any point) and iterates it a few times, with the return map, until it is likely that the iterate is close to the attractor. The box that contains this iterate will most likely also contain points from the return map attractor and we therefore use this box as the starting box in a covering of the attractor.

**Step 3 and 4**

In step 3 and 4 we determine inside which boxes the starting box from step 1 is mapped with the return map. The boxes hit by the starting box are then put through the same process and their hit boxes are found. By repeating this for all new boxes added to the covering we eventually get a collection of boxes which is mapped inside itself. This is a covering of the attractor.

**Step 5 and 6**

In the Poincaré plane there is a discontinuity of the return map caused by points flowing on different sides of the stable manifold of the origin. We find the boxes in the covering that cross the discontinuity, and pick a small cube centred in the origin having faces parallel to the coordinate planes. The discontinuity boxes are then mapped with the flow stepwise down to the top of the cube and back to the return plane. Orbits flowing close to the origin are thereby split up into smaller pieces to avoid certain numerical problems associated with the great expansion of these orbits.

If any of the boxes hit by the returning discontinuity boxes are new to the covering they are added to the covering and we run step 3 and 4 again to make the covering invariant.

**Step 7**

After step 3 to 6 are done we have an invariant collection of boxes and inside this box covering there is an attractor of the return map. Furthermore, in step 3 to 6 we have also calculated a value of the differential map for each mapping from one
2.1. A SHORT SUMMARY

Figure 2.2: Points in the boxes crossing the discontinuity ➊ flow stepwise down to a cube centred around the origin, ➋ flow through the cube via an analytical coordinate change and then ➌ flow back to the Poincaré plane.

box to another box. This means that if box $i$ is mapped partially over box $j$ then we have calculated the value of the differential in a point in box $i$ which is mapped to a point inside box $j$. These values of the differential map are used, in a first phase, as a piecewise constant approximation of the differential map.

The object of step 7 to 9 is to construct inner and outer cones, on the boxes of the covering, that are invariant when mapped with this piecewise constant approximation of the differential map. Step 7 creates inner and outer starting cones on a small sub-collection of the boxes to prime step 8 and 9, where the invariant cones under this approximation of the differential are created.

**Step 8 and 9**

We take the cones created in step 7 and push them forward. This means that if the cone pair on box $i$ is mapped to a box $j$ then we create an inner and outer cone on box $j$ that contain the image of the outer cone of box $i$. If box $j$ already has a pair of cones then we instead check that they contain the image cone, and, if not, we enlarge them. Anytime a new or changed cone appears it is pushed forward in this

Figure 2.3: After step 7 has been run an initial set of cones are defined on a sub-collection of boxes.
way. This is repeated until the cones have reached an equilibrium, at which point the cones are invariant (each inner cone contains all images of outer cones being mapped to this cone).

**Step 10 and 11**

The result of step 8 and 9 is a collection of cones that are invariant under the piecewise constant approximation of the differential map. Since we expect this approximation of the differential to be fairly accurate these cone should also be close to invariant under the (non-constant) differential map.

In step 10 and 11 we use the cones from step 8 and 9 as starting cones when we generate invariant cones under the proper differential map by applying the same algorithm as in step 8 and 9 (push cones forward).

**Step 12 and 13**

After having created the invariant cones we determine the smallest expansion inside the outer cone of the starting box in each mapping between two boxes.

**Step 14 and 15**

At this point we can create a graph that captures the mapping and expansion information of the return map on the box covering. In this graph the nodes correspond to the boxes and a directed edge from node $i$ to node $j$ with weight $w$ means that box $i$ is mapped to box $j$ and that the logarithm of the smallest expansion factor of this mapping is $w$.

The problem of showing that the return map attractor is a strange attractor can be translated into showing two properties of this graph:

- Every cycle in the graph has positive weight.
- Every path between two nodes that correspond to discontinuity boxes (mentioned in step 5 and 6) has weight greater than $\log 2$. 

**Figure 2.4:** Invariant cones means that if box $i$ is mapped to box $j$ then the outer cone on box $i$ is mapped with the differential map inside the inner cone on box $j$. 

inner cone $i$ outer cone $i$  

$dR$  

inner cone $j$ outer cone $j$  

$dR$ (outer cone $i$)
2.2. **STEP 0**

The first property is checked in step 14 which uses an exterior graph analysing program to investigate if there are any negative cycles. Step 15 verifies the second property, also using an exterior graph analysing program.

### 2.2 Step 0

Step 0 is really just a preliminary step. It consists of some Maple programs whose task is to generate a number of important numerical Matlab routines. The reason for using an indirect way of creating the Matlab functions is that due to their complexity they would be very tedious or next to impossible to write correctly by hand, and also that we can take advantage of the code generating capabilities of Maple to make the generated Matlab functions more efficient than would otherwise be possible.

A sum-up of the five Maple programs is:

- **lsodedata.mpl** outputs two parameter files for the Lorenz-Stenflo differential equations (after the linear change of coordinates).
- **R.mpl** creates the return map routine.
- **box_R.mpl** is a variation of **R.mpl** and creates an integrator routine for orbits confined to a box. This routine will be used when we approximate orbits that approach the origin.
- **xphi.mpl** and **xpsi.mpl** generate the coordinate change routines around the origin.

Below we make a short presentation of each of these programs and the Matlab functions they generate.

### A linear change of coordinates

One of the first things we do when looking at the Lorenz-Stenflo system is to make a linear change of coordinates so that the stable and unstable directions of the origin coincide with the new coordinate axes. This coordinate change takes the form

\[
\begin{pmatrix}
x \\
y \\
z \\
v
\end{pmatrix}
= \begin{pmatrix}
1 & 1 & 1 & 0 \\
\alpha_1 & \alpha_2 & \alpha_3 & 0 \\
0 & 0 & 0 & 1 \\
\gamma_1 & \gamma_2 & \gamma_3 & 0
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{pmatrix},
\]

where

\[
\alpha_i = \frac{\lambda_i^2 + 2\sigma\lambda_i + \sigma^2 + s}{\sigma(\lambda_i + \sigma)}, \quad \gamma_i = -\frac{1}{\lambda_i + \sigma}
\]

and \(\lambda_1 > 0 > \lambda_2 > \lambda_3, \lambda_4 = -\beta\) are the eigenvalues of the linearised system in the origin. In this new coordinate system the \(x_4\)-direction coincides with the \(z\)-direction and the \(x_1\)-direction is the direction of the unstable manifold of the origin.
CHAPTER 2. THE NUMERICAL METHOD

**Figure 2.5:** A schematic outline of how all steps are interrelated and communicate via data files.
2.2. STEP 0

Figure 2.5: (Cont.)
After this change of coordinates the Lorenz-Stenflo system is transformed into a system where the linear part is in a diagonal form with the eigenvalues $\lambda_1, \ldots, \lambda_4$ as diagonal elements and where the second order terms are merely rearranged. With new constants $k_1, \ldots, k_9$ the system can be written as

$$\begin{aligned}
\dot{x}_1 &= \lambda_1 x_1 + k_1 (x_1 + x_2 + x_3) x_4 \\
\dot{x}_2 &= \lambda_2 x_2 + k_2 (x_1 + x_2 + x_3) x_4 \\
\dot{x}_3 &= \lambda_3 x_3 + k_3 (x_1 + x_2 + x_3) x_4 \\
\dot{x}_4 &= \lambda_4 x_4 + k_4 x_1^2 + k_5 x_1 x_2 + k_6 x_1 x_3 + k_7 x_2^2 + k_8 x_2 x_3 + k_9 x_3^2
\end{aligned}$$

or in an abbreviated form $\dot{x} = Ax + F(x)$.

In the Maple program lsodedata.mpl we calculate the numerical values of the constants $\lambda_1, \ldots, \lambda_4$ and $k_1, \ldots, k_9$, and output them in the file lsodedata.m.

The return map

In an attempt to simplify our study of the Lorenz-Stenflo system we have introduced a Poincaré plane $x_4 = \rho - 1 - s/\sigma^2$ and a return map $R$ that maps a point $x$ in the Poincaré plane to the point where the orbit of $x$ hits this plane again, with downward pointing direction. The dynamical properties of the whole system then translate into dynamical properties of the return map on the Poincaré plane.

Calculating the return map amounts to integrating a point according to the Lorenz-Stenflo system

$$\frac{d}{dt} \varphi_i(x, t) = f_i(\varphi(x, t))$$

until it hits the Poincaré plane. In this equation $\varphi(x, t)$ denotes the flow and is the point that corresponds to time $t$ on the orbit that starts in $x$ at time 0.

Since we are also interested in calculating the derivatives of the return map we need the first and second order derivatives $\partial \varphi_i/\partial x_j$ and $\partial^2 \varphi_i/\partial x_k \partial x_j$ of the flow. We find equations for these derivatives by differentiating both sides of the Lorenz-Stenflo system, and thus deriving the so-called equations of first and second variation.
\[ \frac{d}{dt} \frac{\partial \varphi_i}{\partial x_j}(x, t) = \frac{\partial}{\partial x_j} f_i(\varphi(x, t)) = \sum_\ell \frac{\partial f_i}{\partial x_\ell}(\varphi(x, t)) \frac{\partial \varphi_\ell}{\partial x_j}(x, t), \]

\[ \frac{d}{dt} \frac{\partial^2 \varphi_i}{\partial x_k \partial x_j}(x, t) = \frac{\partial}{\partial x_k} \sum_\ell \frac{\partial f_i}{\partial x_\ell}(\varphi(x, t)) \frac{\partial \varphi_\ell}{\partial x_j}(x, t) \]
\[ = \sum_{\ell, n} \frac{\partial^2 f_i}{\partial x_n \partial x_\ell}(\varphi(x, t)) \frac{\partial \varphi_n}{\partial x_k}(x, t) \frac{\partial \varphi_\ell}{\partial x_j}(x, t) \]
\[ + \sum_\ell \frac{\partial f_i}{\partial x_\ell}(\varphi(x, t)) \frac{\partial^2 \varphi_\ell}{\partial x_k \partial x_j}(x, t). \]

The return map can now be defined as

\[ R_i(x) = \varphi_i(x, T(x)) \quad \text{for } i = 1, 2 \text{ and } 3, \]

where \( T(x) \) denotes the time it takes for the point \( x \) to flow back to the Poincaré plane. The time \( T(x) \) is therefore defined implicitly by the equation \( \varphi_4(x, T(x)) \equiv C \).

In order to calculate the derivatives of the return map we first differentiate both sides of \( \varphi_4(x, T(x)) \equiv C \) to get an expression for the derivatives of \( T(x) \),

\[ \frac{\partial}{\partial x_j} \varphi_4(x, T(x)) = \frac{\partial \varphi_4}{\partial x_j}(x, T) + \dot{\varphi}_4(x, T) \frac{\partial T}{\partial x_j}(x) \]
\[ = \frac{\partial \varphi_4}{\partial x_j}(x, T) + f_4(\varphi(x, T)) \frac{\partial T}{\partial x_j}(x) = 0 \]
\[ \Leftrightarrow \frac{\partial T}{\partial x_j}(x) = -\frac{1}{f_4(\varphi(x, T))} \frac{\partial \varphi_4}{\partial x_j}(x, T). \]

We can then calculate the derivatives of \( R \),

\[ \frac{\partial R_i}{\partial x_j}(x) = \frac{\partial}{\partial x_j} \varphi_i(x, T(x)) = \frac{\partial \varphi_i}{\partial x_j}(x, T) + \dot{\varphi}_i(x, T) \frac{\partial T}{\partial x_j}(x) \]
\[ = \frac{\partial \varphi_i}{\partial x_j}(x, T) - \frac{f_i(\varphi(x, T))}{f_4(\varphi(x, T))} \frac{\partial \varphi_4}{\partial x_j}(x, T). \]

Strictly speaking we have no direct use for the second order derivatives of the return map but as they turn out to be useful in certain situations (mostly related to improving some optimisation routines) we will nevertheless include them in the return map calculations. The second order derivatives are obtained by differentiating the
first order derivatives again,

\[
\frac{\partial^2 R_i}{\partial x_k \partial x_j}(x) = \frac{\partial}{\partial x_k} \left( \frac{\partial \varphi_i}{\partial x_j}(x, T) - \frac{f_i(\varphi(x, T)) \partial \varphi_4}{f_4(\varphi(x, T)) \partial x_j}(x, T) \right)
\]

\[
= \frac{\partial^2 \varphi_i}{\partial x_k \partial x_j}(x, T) + \left( \frac{d}{dt} \frac{\partial \varphi_i}{\partial x_j}(x, T) \right) \frac{\partial T}{\partial x_k}(x) - g_{ik} \cdot \frac{\partial \varphi_4}{\partial x_j}(x, T)
\]

\[
- \frac{f_i(\varphi(x, T))}{f_4(\varphi(x, T))} \left( \frac{\partial^2 \varphi_4}{\partial x_k \partial x_j}(x, T) + \left( \frac{d}{dt} \frac{\partial \varphi_4}{\partial x_j}(x, T) \right) \frac{\partial T}{\partial x_k}(x) \right),
\]

where \(g_{ik}\) denotes the \(x_k\)-derivative of the quotient \(f_i(\varphi(x, T))/f_4(\varphi(x, T))\) and is equal to

\[
g_{ik} = \frac{1}{f_4(\varphi(x, T))^2} \left[ \left( \sum_{\ell=1}^{4} \frac{\partial f_i}{\partial x_\ell}(\varphi(x, T)) \cdot \frac{\partial R_\ell}{\partial x_k}(x) \right) \cdot f_4(\varphi(x, T)) \right]
\]

\[
- f_i(\varphi(x, T)) \cdot \left( \sum_{\ell=1}^{4} \frac{\partial f_4}{\partial x_\ell}(\varphi(x, T)) \cdot \frac{\partial R_\ell}{\partial x_k}(x) \right). \]

It is now clear how the Matlab return map routine must work. It integrates the flow variables \(\varphi_i\) and its derivatives \(\partial \varphi_i/\partial x_j\) and \(\partial^2 \varphi_i/\partial x_k \partial x_j\) until time \(t = T(x)\), after which it calculates the return map together with its derivatives according to the formulas we derived above.

The Maple program \texttt{R.mpl} derives all the needed equations and formulas, and then generates the main return map routine \texttt{R.m}. This Matlab routine has the following function prototype

\[
[Y, DY, DDY] = R(X);
\]

where the output arguments encode the return map information in the following way

\[
Y(i) = R_i(X), \quad DY(i,j) = \frac{\partial R_i}{\partial x_j}(X), \quad DDY(i,j,k) = \frac{\partial^2 R_i}{\partial x_k \partial x_j}(X).
\]

Normally \(X\) has three components and is assumed to be a point in the Poincaré plane, but if it has four components then it is instead interpreted as a point in the phase space.

The box integrator routine

Besides the return map routine we also make use of a Matlab routine \texttt{box_R.m} that doesn’t flow points back to the Poincaré plane but restricts orbits to the inside of a box. This routine will later on be used to subdivide certain difficult orbits into more manageable pieces. Its function prototype is
[HT, Y, DY, D DY] = box_R (X, XMIN, XMAX);

where X is a point inside the box XMIN(i) ≤ x_i ≤ XMAX(i). The output Y is the point on the boundary of the box where the orbit of X exits the box, and the output variables DY and D DY are the first and second order derivatives, respectively, of the mapping from X to Y. The first output variable HT indicates on which boundary face of the box the orbit of X exits the box. If Y(i) = XMIN(i) then HT(i) = -1 and if Y(i) = XMAX(i) then HT(i) = 1. Otherwise, if XMIN(i) < Y(i) < XMAX(i) then HT(i) = 0.

The function box_R.m is to a large extent similar to the return map routine and is based on the same type of calculations as that function.

The coordinate change functions

Around the origin we use a change of coordinates that almost linearises the flow to help us analyse those orbits that flow very close to the origin. This coordinate change x = φ(z) and its inverse z = ψ(x) are derived in Chapter 3 and expressed as power series

φ_i(z) = \sum_j c_{i,j} z^j \quad \text{and} \quad \psi_i(x) = \sum_j e_{i,j} x^j,

where the coefficients c_{i,j} and e_{i,j} are shown to satisfy certain recurrence equations.

In the Maple programs xphi.mpl and xpsi.mpl we use the recurrence equations to calculate all coefficients up to a certain degree and output Matlab routines xphi.m and xpsi.m, respectively, that approximate φ and ψ by truncated power series. The function prototype of xphi.m and xpsi.m are

[X, DX, DDX] = xphi(Z) \quad \text{and} \quad [Z, DZ, DDZ] = xpsi(X),
where the output arguments are

\[
X(i) = \phi_i(Z), \quad DX(i,j) = \frac{\partial \phi_i}{\partial z_j}(Z), \quad DDX(i,j,k) = \frac{\partial^2 \phi_i}{\partial z_k \partial z_j}(Z),
\]

\[
Z(i) = \psi_i(X), \quad DZ(i,j) = \frac{\partial \psi_i}{\partial x_j}(X), \quad DDZ(i,j,k) = \frac{\partial^2 \psi_i}{\partial x_k \partial x_j}(X).
\]

### 2.3 Step 1 and 2

The main objective of these two steps is to prepare for step 3 and 4. We start by explaining some Matlab functions related to the box covering and which will also be used in later steps before we go through the programs step01.m and step02.m.

#### A subdivision of the Poincaré plane

The Poincaré plane is subdivided into a grid of small boxes of equal side length \(dx\),

\[
B_{i,j,k} = \{ (x_1, x_2, x_3) : \ i \ dx \leq x_1 \leq (i + 1) \ dx, \ j \ dx \leq x_2 \leq (j + 1) \ dx, \ k \ dx \leq x_3 \leq (k + 1) \ dx \}
\]

and a subcollection of these boxes will in later steps be used to cover the whole return map attractor.

When we work with the boxes in this subdivision we will keep track of individual boxes by their indices and to facilitate converting back and forth between boxes and coordinates there are a couple of functions which we use throughout all steps.

- \(B = \text{to\_box}(P)\). This function outputs the index vector \(B\) of the box that contains the point \(P\).

- \([\text{XMIN}, \text{XMAX}] = \text{from\_box}(B)\). The input \(B\) is the index vector of a box and the output \(\text{XMIN}\) and \(\text{XMAX}\) specify the coordinate range of that box, i.e. \(\text{XMIN}(i) \leq x_i \leq \text{XMAX}(i)\).

In an effort to cut down on the workload in subsequent steps we will only cover one branch of the return map attractor with boxes and work exclusively with these boxes. This doesn’t really leave anything out since by using the symmetry of the return map, \(R(-x_1, -x_2, -x_3) = -R(x_1, x_2, x_3)\), we can thereafter induce a symmetric box covering of the other branch and also translate mapping properties to those boxes.

We therefore split the Poincaré plane in two halves \(\{x_2 \geq -\frac{1}{6} \ x_1\}\) and \(\{x_2 \leq -\frac{1}{6} \ x_1\}\), where we have assured that each half contains one of the branches. The mapping between the two symmetric halves is taken care of by the following functions.
2.3. \textit{STEP 1 AND 2}

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{box_covering.png}
\caption{The box covering has two components, each covering one branch of the return map attractor. These components are symmetric and on different sides of the plane $x_2 = -\frac{1}{6}x_1$.}
\end{figure}

- $Y = x\sigma(X)$. This function maps a point $X$ to its symmetric point in the other half of the Poincaré plane.

- $M = dx\sigma$. The returned matrix $M$ is the differential of the symmetry mapping.

- $S = x\sigma\_box(B)$. This function is analogous to $x\sigma$ and maps a box with index vector $B$ to its symmetric box, which has index vector $S$.

We also have corresponding functions $x\text{sym}$, $dx\text{sym}$ and $x\text{sym}\_box$ that are equal to $x\sigma$, $dx\sigma$ and $x\sigma\_box$, respectively, on the half \{\(x_2 \leq -\frac{1}{6}x_2\)\} but equal to the identity on the other half \{\(x_2 \geq -\frac{1}{6}x_1\)\}. In this way these functions always map to the second half of the Poincaré plane; the half where the return map will be studied.

\textbf{step01.m}

The task of the Matlab program \texttt{step01.m} is to find one starting box for the box covering of the attractor. We begin by picking a point $x$ in the Poincaré plane and iterate it a few times with the return map. The resulting point will most likely be close to the attractor and as a starting box we choose the box that contains this point (or the symmetric box if this box is not in the correct half of the Poincaré plane). This box is then output in the file \texttt{unfinished.dat} in the format

\begin{verbatim}
i j k
\end{verbatim}

where \((i,j,k)\) is the index of the box. We initialise the data files \texttt{variables.mat} and \texttt{added\_boxes.mat} with empty variables (these variables are explained later).

\textbf{step02.m}

The next Matlab program \texttt{step02.m} just iterates a point a large number of times to get a collection of points around the return map attractor. Step 4 will use these
points when providing progress reports. The points are stored in the $3 \times N$-matrix \textit{orbit} as columns and this matrix is output in the file \textit{orbit.mat}.

2.4 Step 3 and 4

In step 3 and 4 we will produce a collection of boxes that cover one branch of the return map attractor and where each box in the covering is mapped with the return map inside the covering itself or inside the mirror covering (that covers the other branch of the attractor). This is what is known as an invariant covering of the attractor. When these two steps have finished we will have generated, besides the covering itself, the following mapping information for each box:

- a list of boxes from the covering that together contain the image of the box,
- points in the box that are mapped inside each one of the hit boxes that cover the image,
- the images of these points, and
- the value of the differential map in each of these points.

The whole covering is built up step by step starting with the single box in the file \textit{unfinished.dat} (the result of step 1) and the adding of boxes to the covering is done by the program \textit{step03.m} using the following method: \textit{step03.m} takes a box from the file \textit{unfinished.dat}. For this box the program determines the mapping information mentioned above and outputs it in the file \textit{boxes.dat}. Among the boxes that cover the image of the box there are boxes that have not yet been processed by \textit{step03.m} and these boxes are added to \textit{unfinished.dat}. After that \textit{step03.m} starts over with a new box from \textit{unfinished.dat}. This continues until no more boxes are found in \textit{unfinished.dat}.

The program \textit{step03.m} can be run as several independent processes all working in parallel and to keep order of the boxes in the covering and their status we use three data files

- \textit{unfinished.dat}. This file contains boxes that have been added to the covering but not yet processed by \textit{step03.m}.
- \textit{inprogress.dat}. In this file are the boxes currently being processed by the program \textit{step03.m}.
- \textit{finished.dat}. Boxes in this file have already been processed by \textit{step03.m}.

Alongside \textit{step03.m} we run \textit{step04.m} which continuously reports how the covering is progressing and when \textit{step03.m} has finished, it collects all the data from the file \textit{boxes.dat} and outputs it as variables in the file \textit{variables.mat}. 
A lock mechanism

In order to run step03.m as several parallel processes we have a mechanism to prevent the individual processes from interfering with each other. If a process wants to read or write to any of the files unfinished.dat, inprogress.dat, finished.dat or boxes.dat it creates a file called LOCK which gives the process exclusive right to change any of these files. The process releases its lock on these files by deleting the file LOCK.

- **get_lock.** This function creates an empty file LOCK in the current directory. If this file already exists the function waits until the file is removed before creating it.
  
The details of this need some attention. We must check if the file exists and, if not, create it, all in one step. Otherwise it is possible for the following scenario to occur

  1. Process 1 checks if LOCK exists.
  2. Process 2 checks if LOCK exists.
  3. Process 1 creates LOCK.
  4. Process 2 creates LOCK.

  (This situation is called a race condition.) The way to handle this on a Unix system is to try to link LOCK to an existing file,

  ```
  ln -s EXISTING_FILE LOCK
  ```

  If LOCK already exists this command will fail and if the command succeeds LOCK is created.

  ```
  function get_lock();
  [s,msg] = unix(’ln -n EXISTING_FILE LOCK’);
  while s~=0
    pause(5);
    [s,msg] = unix(’ln -n EXISTING_FILE LOCK’);
  end;
  ```

- **rm_lock.** This function deletes the file LOCK.

  ```
  function rm_lock();
  unix(’rm -rf LOCK’);
  ```

**step03.m**

We have already discussed the basic tactics of step03.m and now we give a proper rundown of the program. The general structure of the program can be outlined in the following way:
CHAPTER 2. THE NUMERICAL METHOD

⟨Initialisation⟩
⟨Get an unfinished box⟩

while ~done
   ⟨Store the boxes hit by the corner points of box in HB⟩
   ⟨Find all boxes hit by box⟩
   ⟨Output box data⟩
   ⟨Get a new unfinished box⟩
endwhile

The rest of this section is devoted to explaining the different code sections found in the outline above.

⟨Initialisation⟩

All Matlab functions are kept in the directory ../m-files so we add the full path of this directory to the Matlab search path.

28 clear;
29 p = pwd; cd('..'); parent=pwd; cd(p);
30 path(path,[parent '/m-files']);
31 lsdata;
32 lsodedata;

In the data file lsdata.m the following constants are defined:

    d    The dimension of the Poincaré plane.
    dx   Side length of the boxes in the Poincaré plane.
    excess An amount added to cones (see step 8 and 9).
    interplanes A row vector of $x_4$-coordinates of all intermediate planes between the Poincaré plane and the origin cube (see step 5 and 6).
    cube_radius Half of the side length of the origin cube.

and in lsodedata.m:

    lambda The linear coefficients of the Lorenz-Stenflo system.
    k     The quadratic coefficients of the Lorenz-Stenflo system.
    pplane The $x_4$-coordinate of the Poincaré plane.

⟨Get an unfinished box⟩

We get a box from unfinished.dat by using get_unfinished_box.

34 box = get_unfinished_box;
35 done = 0;
36 if size(box,2)==0, done=1; end;
This function either returns the index vector of a box or an empty matrix if there
are no remaining boxes in unfinished.dat and inprogress.dat is empty (a box in
inprogress.dat could potentially generate new boxes in unfinished.dat so both
these files need to be empty before we know that all boxes have been processed).

(Store the boxes hit by the corner points of box in HB)

The goal in this program is to find all boxes that intersect the image of box. This
will be done in a process which starts with a small number of boxes we know are
hit by box, and then this set of boxes is enlarged by adding one box at a time.

All the boxes hit by box and related information are kept throughout the pro-
gram in the following matrices:

- HB. The columns of this matrix contains the indices of the boxes hit by box.
- SP. Each column of this matrix is a point in box that is mapped to the
  corresponding box in HB.
- HP. The image points of the points in SP are columns of this matrix.
- D. The differentials of the return map at the points in SP are stored as columns
  in this matrix.

We initialise these variables by mapping the corner points of box and determining
which boxes are hit by the points.

% Map the corners of the box
[xmin,xmax] = from_box(box);
oxmin = [xmin; pplane];
oxmax = [xmax; pplane];
p = cube_corners(xmin,xmax);
q = zeros(d,pow2(d));
dq = zeros(d*(d+1),pow2(d));
for i=1:pow2(d)
    [q(:,i),tmp] = R(p(:,i));
dq(:,i) = tmp(:);
end;
[HB,SP,HP,D] = delete_dup(to_box(q),p,q,dq);

In the lines above we have used the following functions

- cube_corners(xmin,xmax). The output of this function is the corner points
  of the cube xmin(i) ≤ xi ≤ xmax(i).
- delete_dup(a,b,c,...). This function removes duplicate columns in a
  and the corresponding columns that are removed from a are also removed
  from b,c,...
CHAPTER 2. THE NUMERICAL METHOD

〈Find all boxes hit by box〉

The work of finding all boxes in the Poincaré plane intersecting the image of box is done by the function `find_hit_boxes`.

```matlab
% Find all hit boxes
[HB,SP,HP,D] = find_hit_boxes(xmin,xmax,zeros(1,d+1),...  
0,HB,SP,HP,D,1,1);
```

This function has the following function prototype

```
[B,X,Y,D] = find_hit_boxes(XMIN,XMAX,N,C,B,X,Y,D,OB,FID);
```

where the columns of the output matrix B are index vectors of all boxes hit by points from the box \(XMIN(i) \leq x_i \leq XMAX(i)\). (The input arguments N and C are not used in this program.) In the column \(X(:,I)\) of the output matrix X there is a point from the starting box \([XMIN,XMAX]\) which is mapped to a point \(Y(:,I)\) inside the box \(B(:,I)\). The differential of this mapping is stored as the column \(D(:,I)\) (and the command `reshape` restores the matrix). To set up the search for hit boxes the function uses the initial data found in the input arguments \(B, X, Y\) and \(D\).

If \(OB = 1\) then those boxes that are added to \(B\) that are not yet part of the covering (i.e. not in `unfinished.dat`, `inprogress.dat` nor in `finished.dat`) are also added to the covering (i.e. to `unfinished.dat`).

Since the function `find_hit_boxes` often takes a long time to finish it also outputs progress information in the file with file identifier `FID` (standard output if `FID=1` or no output if `FID=0`).

In short, the algorithm in `find_hit_boxes` dynamically expands the set of boxes found in the input argument \(B\). The key step in this enlargement is to look at neighbouring boxes to the boxes in \(B\) and investigate if any of them is hit by points from the box \([XMIN,XMAX]\). Such boxes are added to \(B\) and in the next iteration their neighbouring boxes are investigated in the same way. This goes on until no neighbours to the boxes in \(B\) intersect the image of the starting box. At that time we will have found all boxes hit by the starting box.

This algorithm also works when the return map has a discontinuity in the starting box (under the assumption that not all points in the input matrix \(X\) are on the same side of the discontinuity) but this situation also involves numerical problems and will be treated further in step 5 and 6.

Inside the function `find_hit_boxes` the problem of determining if the image of the starting box \([XMIN,XMAX]\) intersects a box \(b\) (a neighbour to the boxes in \(B\)) is solved by a search for a point in the starting box which is mapped inside \(b\). In order for this search to be successful and swift it is necessary to find a suitable starting guess and this is done in the following way:

1. Take all boxes \(b_1,\ldots, b_n\) in \(B\) that are neighbours to \(b\) (in the function these boxes are stored in the variable `nboxes`).
2.4. **STEP 3 AND 4**

Figure 2.8: This figure illustrates how the starting guess in the function `find_hit_boxes` is found, as outlined in the text.

2. Since $b_1, \ldots, b_n$ are in $B$ there are points $p_1, \ldots, p_n$ (which we store in `sguess`) in the starting box that are mapped to points $q_1, \ldots, q_n$ (stored in `iguess`) inside the boxes $b_1, \ldots, b_n$, respectively.

3. Find the convex combination $q = \alpha_1 q_1 + \cdots + \alpha_n q_n$, where $0 \leq \alpha_i \leq 1$ and $\alpha_1 + \cdots + \alpha_n = 1$, that is closest to the box $b$. The corresponding convex combination $p = \alpha_1 p_1 + \cdots + \alpha_n p_n$ will be our starting guess of a point in the starting box that hits $b$.

This scheme is implemented in the function `closest_point` which outputs the point $p$ and the point in $b$ closest to $q$ (which we use as a starting guess in box $b$ for the image point of $p$).

The actual search for a point being mapped to box $b$ is done with the function `intersection` which has the following function prototype,

$$[X, Y, D, I] = \text{intersection}(\text{XMIN}, \text{XMAX}, N, C, \text{YMIN}, \text{YMAX}, \text{SG}, \text{HG});$$

with input and output arguments

- $[\text{XMIN}, \text{XMAX}]$ are the bounds of the starting box and $[\text{YMIN}, \text{YMAX}]$ are bounds of the hit box ($b$ in our case). The point $\text{SG}$ in $[\text{XMIN}, \text{XMAX}]$ and the point $\text{HG}$ in the box $[\text{YMIN}, \text{YMAX}]$ are starting guesses for the search.

- The output argument $I$ is equal to 1 if there is a point $X$ in $[\text{XMIN}, \text{XMAX}]$ that is mapped to a point $Y$ in $[\text{YMIN}, \text{YMAX}]$, otherwise $I$ is equal to 0. The matrix $D$ is the differential of the return map at $X$.

The algorithm in `intersection` works by solving an optimisation problem. To start with, we have starting guesses $X = \text{SG}$ and $Y = \text{HG}$, and typically $R(X) \neq Y$. This non-equality can be rewritten as $Y_i - R_i(X) \pm \mu_i = 0$, where $\mu_i$ measures the discrepancy between $Y_i$ and $R_i(X)$, and the sign $\pm$ is chosen so that $\mu_i \geq 0$. 

CHAPTER 2. THE NUMERICAL METHOD

Now we solve the minimisation problem

\[
\begin{align*}
\min & \quad \mu_1 + \mu_2 + \mu_3 \\
\text{s.t.} & \quad Y_i - R_i(X) \pm \mu_i = 0, \\
& \quad X \in [X_{\text{MIN}}, X_{\text{MAX}}], \\
& \quad Y \in [Y_{\text{MIN}}, Y_{\text{MAX}}], \\
& \quad \mu_i \geq 0.
\end{align*}
\]

If this problem has solution \( \mu_1 + \mu_2 + \mu_3 = 0 \) then we must have \( \mu_1 = \mu_2 = \mu_3 = 0 \) and the corresponding points \( X \) and \( Y \) satisfy \( Y = R(X) \). Otherwise, if \( \mu_1 + \mu_2 + \mu_3 > 0 \), there is no point \( X \) in \([X_{\text{MIN}}, X_{\text{MAX}}]\) being mapped to a point \( Y \) in \( HB \).

In some infrequent cases the optimisation routine in Matlab is unable to solve the minimisation problem and an error message is output in the file \texttt{error.log}. This happens exclusively when the starting box crosses the discontinuity of the return map and this case is studied in more detail in step 5 and 6.

\section*{Output box data}

Once we are finished with box we output all the data found in \( HB \) and related variables. At the same time box is moved from \texttt{inprogress.dat} to \texttt{finished.dat}.

\begin{verbatim}
output_box_data(0,0,box,HB,SP,HP,D(1:d*d,:));
move_box(box,’inprogress.dat’,’finished.dat’);
\end{verbatim}

\section*{Get a new unfinished box}

The last thing we do is to get a new box and start all over.

\begin{verbatim}
box = get_unfinished_box;
if size(box,2)==0, done=1; end;
\end{verbatim}

\section*{step04.m}

The program \texttt{step04.m} has a fairly simple purpose. First it is to report on how the program \texttt{step03.m} is progressing, then to rearrange the mapping data output by \texttt{step03.m} in a more suitable form and lastly to prepare for step 5 and 6.

\section*{Initialisation}

\section*{Track the progress made by \texttt{step03.m}}

\section*{Save box data in \texttt{variables.mat}}

\section*{Save new boxes crossing the discontinuity in \texttt{unfinished.dat}}
2.4. **STEP 3 AND 4**

(Track the progress made by step03.m)

Once every minute we track the progress made by `step03.m` by plotting the return map attractor (the points from `step02.m`) and the boxes produced so far using the following colour scheme:

- **red** for boxes in `unfinished.dat`.
- **yellow** for boxes in `inprogress.dat`.
- **black** for boxes in `finished.dat`.

(Save box data in variables.mat)

When `step03.m` has finished the next phase is to store the box information produced by step 3 in `variables.mat`. The data file `boxes.dat` that `step03.m` uses for output has the following seven-line format:

```
i
s
a1 a2 a3
b1 b2 b3
x1 x2 x3 x4
y1 y2 y3
d1 d2 d3 d4 d5 d6 d7 d8 d9
```

where this means that the image of the box with indices \((a1, a2, a3)\) intersects the box with indices \((b1, b2, b3)\) and that the point \((x1, x2, x3)\) in the first box is mapped to \((y1, y2, y3)\) in the second box (the redundant fourth coordinate \(x4\) is only included to enable us to use the same box data format in step 5 and 6). The differential of the return map at \((x1, x2, x3)\) is equal to

\[
\begin{pmatrix}
d1 & d4 & d7 \\
d2 & d5 & d8 \\
d3 & d6 & d9
\end{pmatrix}.
\]

The first two lines \(i\) and \(s\) are only used in step 5 and 6 and are set to zero in this program. All mapping information will be repackaged in the variables listed below and then saved in the file `variables.mat`.

- **BOXES**: Each column of this matrix contains the indices of a box belonging to the covering.

- **CNS**: If the image of box \(i\) intersects box \(j\) (boxes are numbered according to their column index in `BOXES`) then the element \((i, j)\) in this matrix is non-zero. When the index \(j\) is between \(N + 1\) and \(2N\), where \(N\) is the number of boxes in `BOXES`, then the box \(j\) is on the symmetric covering of the other branch of the attractor and is the mirror box of box \(j - N\).
• IPS and OC. We explain these variables in step 6.

• SPS. If \( n = CNS(i,j) \) then the column \( SPS(:,n) \) is a point in box \( i \) that is mapped to box \( j \).

• HPS. The columns of this matrix are the image points of the corresponding columns in SPS.

• DS. The columns of this matrix are the values of the differential map evaluated at the corresponding columns in SPS.

• DISC. The boxes in the covering that cross the discontinuity of the return map are listed as columns in this matrix.

In later steps, variables associated with cones and expansion are also added to the file variables.mat. All variables were initialised in step 1 to empty matrices and we extend them with the mapping data generated in step03.m by using read_box_data.

```matlab
load 'input/variables';
load 'input/added_boxes';
[CH,BOXES,DISC,CNS,SPS,HPS,DS,IPS,OC] = ... read_box_data(BOXES,DISC,CNS,SPS,HPS,DS,IPS,OC);
added_boxes = [added_boxes CH];
```

〈Save new boxes crossing the discontinuity in unfinished.dat〉

In step 5 and 6 we will investigate the boxes that cross the discontinuity of the return map more thoroughly and to prepare for that, we shall extract these boxes and output them in unfinished.dat.

When we run step03.m and step04.m for the first time all boxes in the covering are new to us and produced in this run. It is, however, possible that we are running these two steps a second time (or third) after step 5 and 6. In that situation step05.m and step06.m will have added new boxes to the covering and we are re-running step03.m and step04.m on those boxes to make the covering complete. If this is the case then we are only interested in finding discontinuity boxes among the new boxes (since the old discontinuity boxes have already been through step 5 and 6). To keep track of which boxes are new we use the variable added_boxes.

New discontinuity boxes are found by using find_discontinuity_boxes which searches through the boxes in added_boxes for boxes that are mapped to both branches of the return map attractor (and hence must cross the discontinuity).

```matlab
NEWDISC = find_discontinuity_boxes(added_boxes,BOXES,CNS);
fprintf(1,'\n\n');
if size(NEWDISC,2)~=0
```


2.5 Step 5 and 6

The result of step 3 and 4 is a covering of the return map attractor and in step 5 and 6 we will complement this covering by adding boxes and mapping data that were not properly accounted for in the previous two steps. There is a small fraction of points in the Poincaré plane near the discontinuity of the return map that have orbits flowing close to the hyperbolic fix point in the origin before returning to the Poincaré plane. These orbits are subject to strong expansive and contractive forces which make them elusive to the numerical return map and at the same time important for the overall behaviour of the system.

If we follow one of the boxes, that contains the discontinuity, as it flows down towards the origin, the expansion in the $x_1$-direction and the contraction in the $x_2$, $x_3$-directions will elongate the box in the $x_1$-direction and compress it in the other directions. The result is that most points in the box will simply be deflected
and flow past the origin while a tiny set of points actually comes close to the origin. It will therefore be difficult to isolate the orbits flowing close to the origin. Another complicating fact is that when orbits flow in the vicinity of the origin they can spend an arbitrary long time in a neighbourhood of the origin (depending on how close they are to the stable manifold of the origin) and this must be dealt with in some non-straightforward way since standard numerical ode-routines will break down when used with long integration times.

A way to get a handle of the problematic orbits is implemented in the program step05.m. This program starts with a box in the Poincaré plane that contains the discontinuity (prepared by step04.m) and a small cube $C_{1/10} = \{|x_i| \leq \frac{1}{10}\}$ centred around the origin. It will track how this box flows down to the origin by introducing a number of intermediate planes $\{x_4 = C_i\}$ between the Poincaré plane and the cube. Since every orbit that starts from the box and flows the whole way down to the cube must intersect all intermediate planes inside certain hit regions this means that we will get hold of these close-to-origin orbits if we use the following stepwise flowing method: We let the box flow down and hit the first intermediate plane. In this plane we determine a box that contains that part of the intersection which is inside the hit region. We also fix a plane on each side of the hit region, according to Figure 2.10, and determine boxes in these two planes that intersect the rest of the orbits. These side boxes then flow back to the Poincaré plane and

![Figure 2.9](image-url)

**Figure 2.9:** In a box containing the discontinuity a very narrow ($<10^{-15}$) set of points flow within a distance of $\frac{1}{10}$ of the origin.

![Figure 2.10](image-url)

**Figure 2.10:** A box ① in an intermediate plane flows down and lands in a box ② inside a hit region in the next intermediate plane. The part of the box that flows outside ② lands in two boxes ③ on sloping side planes.
we use the same shooting algorithm as in step 3 to find the boxes in the Poincaré plane hit by these two boxes. The box in the first intermediate plane is used as a new starting box and we let it flow down to the second intermediate plane by repeating the procedure. All this is reiterated until we end up with a box on top of the cube $C_{1/10}$ (which is guaranteed to contain all points of intersection between orbits emanating from the original box and the top lid of the cube).

This puts us on top of the cube. To get around the problem with orbits flowing very close to the origin, making them seemingly untrackable by numerical means, we use a coordinate change. This change of coordinates $z = \psi(x)$, as discussed in Section 1.4 and analysed in Chapter 3, transforms the Lorenz-Stenflo system $\dot{x} = Ax + F(x)$ into an almost linear system $\dot{z} = Az + G(z)$ where the non-linear term $G(z)$ is pushed to be very small close to the stable and unstable manifolds. The orbits of this so-called normal form system can, as an effect, be approximated well by the corresponding orbits of the linear system $\dot{z} = Az$ and these orbits can be integrated analytically, which factors out the problem of long integration times. Once orbits exit the cube at $z_1 = \pm \frac{1}{10}$ we change back to the standard coordinates $x = \phi(z)$ and resume with the numerical integrator routine.

A detailed study of this coordinate change and accompanying approximations can be found in Chapter 3 where it is shown that all errors involved are sufficiently small and can be included in our estimates of boxes and cones. After the box has undergone these coordinate transformations and is transported by the flow through the cube we have two new boxes around each of the boundary faces $x_1 = \pm \frac{1}{10}$ that intersect all orbits stemming from the box on top of the cube as they exit the $z$-cube at $z_1 = \pm \frac{1}{10}$. We then use the shooting algorithm of step 3 to find all boxes in the Poincaré plane hit by orbits starting from these two last boxes. If any of the boxes in the Poincaré plane are not part of the covering we need to add them to the covering and re-run step03.m on them.

The program step05.m can be run as many separate processes (just as we

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2.11}
\caption{Origin passage: 1 change coordinates with $\psi$, 2 flow linearly through the cube, 3 change back to the original coordinates with $\phi$ and 4 flow to the Poincaré plane.}
\end{figure}
did with \texttt{step03.m}) and each process works with one of the discontinuity boxes performing the steps described above. Parallel to \texttt{step05.m} we run \texttt{step06.m} which has a similar function as \texttt{step04.m}. It reports the progress \texttt{step05.m} is making and when \texttt{step05.m} has finished collects all data and outputs it in the file \texttt{variables.mat}. (Any newly introduced boxes are output in \texttt{unfinished.dat}.)

\textbf{step05.m}

Before going into the details of the program \texttt{step05.m} we first present it in an outline form in order to give the general plan of the program.

\begin{verbatim}
⟨ Initialization ⟩
⟨ Get a discontinuity box ⟩
while ~done
    % Let box stepwise climb down to the cube
    (Let box be the starting box)
    for \texttt{yd = interplanes}
        ⟨ Define a hit region in the next plane ⟩
        ⟨ Determine a hit box containing the image of the starting box ⟩
        ⟨ Determine a hit box in the left-hand plane ⟩
        ⟨ Determine a hit box in the right-hand plane ⟩
        ⟨ Let the hit box in the next plane be the new starting box ⟩
    endfor
    % Flow the box back to the Poincaré plane
    ⟨ Transform the box to new coordinates with \( \psi \) ⟩
    for \( S = [-1 1] \) % Flow on both sides of the origin
        ⟨ Let the box flow through the origin cube ⟩
        ⟨ Transform back to the original coordinates with \( \phi \) ⟩
        ⟨ Find all boxes in the Poincaré plane hit by the box ⟩
    endfor
    ⟨ Get a new discontinuity box ⟩
endwhile
\end{verbatim}

While reading the rest of this section it is good to keep a print-out of the program next to you, so as not to get lost in all details.

⟨ Get a discontinuity box ⟩

In this program we use the same three-file system as we did in \texttt{step03.m} to keep track of boxes and their status. This time the file \texttt{unfinished.dat} carries only boxes that intersect the discontinuity of the return map.
2.5. **STEP 5 AND 6**

```matlab
done = 0;
box = get_unfinished_box;
if size(box,2)==0, done=1; end;
```

**Let box be the starting box**

Throughout the descent to the cube we will let \([\text{xmin}, \text{xmax}]\) be the coordinate range of the starting box, which we initialise to be \(\text{box}\). In every mapping from one intermediate plane to the next we will also calculate the differential of that mapping (in some point in the first plane) and the variable \(D\) will be the differential accumulated so far by multiplying the differentials of these stepwise mappings (\(D\) will eventually form an approximate differential of the whole mapping from the box down to the cube).

```matlab
[xmin,xmax] = from_box(box); xmin = [xmin; pplane]; xmax = [xmax; pplane]; D = eye(d+1); k = 0;
```

**Define a hit region in the next plane**

Our first task is to choose a hit region in the next intermediate plane that contains all points of intersection between the plane and orbits flowing from the starting box down to the cube.

If we look at the set of all intersection points in the next plane with orbits originating from the starting box (including orbits that do not approach the origin), this set is delimited by orbits that have a tangential intersection with the plane (as illustrated in Figure 2.12(a)). Since these orbits, and nearby orbits, do not flow towards the cube at the origin we shall pick the hit region well within these peripheral points. At the tangential intersection points, the fourth component of the vector field vanishes, which means that

\[
F_4(x) = \lambda_4 x_4 + k_4 x_1^2 + k_5 x_1 x_2 + k_6 x_1 x_3 + k_7 x_2^2 + k_8 x_2 x_3 + k_9 x_3^2 = 0.
\]

For a fixed value of \(x_4\) this describes a second degree algebraic surface in the plane and rewriting it in canonical form makes it easy to read off the shortest distance from the set to the \(x_4\)-axis. We choose the hit region to be a square box centred at the \(x_4\)-axis with side length half of that distance (but not larger than 1).

```matlab
% Pick a hit region in the next plane
fprintf(1,' Next hit region');
if yd ~= interplanes(end)
    [hmin,hmax] = hit_region(yd,1);
else
    hmin = [repmat(-yd,d,1); yd];
```
Figure 2.12: (a) The strong hyperbolicity of the origin deflects orbits from the starting box making them intersect the next intermediate plane tangentially. (b) On the three faces indicated in the figure the vector field points outwards and on the rest of the faces it points inwards.

To rule out that an orbit landing outside the hit region could still flow down to the origin cube we check that the vector field points outwards on the three planar faces illustrated in Figure 2.12(b) and orbits can therefore not enter a corresponding region further down in the $x_1$-direction.

\[
hmax = [\text{repmat}(yd,d,1); \ yd];
\]

\[
\text{end};
\]

\[
\text{BMIN} = \min([\text{xmin(1:end-1)} \ hmin(1:end-1)]-0.05,\ ];
\]
\[
\text{BMAX} = \max([\text{xmax(1:end-1)} \ hmax(1:end-1)]+0.05,\ ];
\]

\[
\text{if k>1}
\]
\[
\text{CouldOrbitReenter = ~check_flowbox(BMIN,BMAX)};
\]
\[
\text{if CouldOrbitReenter}
\]
\[
\text{error('Hit region allows re-entering orbits');}
\]
\[
\text{end};
\]
\[
\text{end;}
\]

(Determine a hit box containing the image of the starting box)

We will now go on to determine a box in the hit region that the orbits from the starting box land inside, but before that we need to find a point $x$ in the starting box that is mapped to a point $y$ in the hit region. This is done with the function \text{stepwise\_intersection}.

The box in the hit region is determined by solving the following optimisation
2.5. **STEP 5 AND 6**

**Figure 2.13:** The function *stepwise_intersection* works by first enclosing the starting box and the hit region in a box, ① map a point in the starting box with *box_R* and ② use shooting to find a point hitting the next plane. Then in a second phase ③ use shooting to find a point landing in the hit region.

Problems

\[
\min/\max \ box_R(x)
\]

\[
\begin{cases}
  x \in \text{starting box}, \\
  box_R(x) \in \text{hit region},
\end{cases}
\]

where *box_R* is used with a box that encloses the starting box and contains the hit region on one of its boundary faces (the same type of box as in Figure 2.13). These problems are solved in the function *stepwise_box*.

```matlab
71  % Determine a middle box
72  [x,y,dd,isHit] = stepwise_intersection(xmin,xmax,hmin,hmax);
73  if ~isHit, error('Unable to find intersection'); end;
74  [ymin,ymax] = stepwise_box(xmin,xmax,hmin,hmax,x,y);
75  output_iplane_data(k,0,box,ymin,ymax,zeros(1,d+1),0);
```

This function has the following function prototype

\[
[YMIN,YMAX] = \text{stepwise}\_\text{box}(XMIN,XMAX,YMIN,YMAX,X,Y);
\]

where the starting box is \(XMIN(i) \leq x_i \leq XMAX(i)\), the hit region \(YMIN(i) \leq x_i \leq YMAX(i)\) and \(X\) is a point in the starting box which lands in a point \(Y\) inside the hit region (these points are used to start the search). The output arguments are the coordinate range of the hit box.

| (Determine a hit box in the left-hand plane) |

Orbits that deviate to the left of the hit region are caught by a plane \(nvec \cdot x = C\) to the left of the hit region.

```matlab
77  % Left-hand plane
78  fprintf(1,'\n Left plane\n');
```
CHAPTER 2. THE NUMERICAL METHOD

if ymin(1)<hmin(1)+1e-5
  \% Pick a left-hand plane
  nvec = [BMIN(d+1)-BMAX(d+1)+0.05 zeros(1,d-1) ... 
  BMIN(1)-hmin(1)];
  C = nvec*hmin;
  CouldOrbitReenter = ~outward_flow(BMIN,BMAX,nvec,C);
  if k~=1 & CouldOrbitReenter
    error('Left hit plane allows re-entering orbits');
  end;

The box in this plane that contains all intersections with the orbits from the
starting box is determined by solving the same type of optimisation problem as
in stepwise_box and this is done in the function stepwise_plane_box.

  \% Determine a left box and hit boxes
  x = cube_corners(xmin,[xmin(1);xmax(2:end)]);
  [ht,y,dy] = box_R(x(:,1),BMIN,BMAX,nvec,C);
  [lmin,lmax] = stepwise_plane_box(xmin,xmax,BMIN,...
  [hmin(1);BMAX(2:end)],BMIN,BMAX,nvec,C,x);
  output_iplane_data(k,-1,box,lmin,lmax,nvec,C);

From the box we use shooting to find which boxes in the Poincaré plane that are
hit.

  SP = y; [HP,DD] = R(SP); DD = DD(:,1); HB = to_box(HP);
  [HB,SP,HP,DD] = find_hit_boxes(lmin,lmax,nvec,C,...
  HB,SP,HP,DD,0,1);
  DD = append_matrix(DD,dy*D);
  output_box_data(k,-1,box,HB,SP,HP,DD(1:d*d,:));
  end;

The code section for the plane on the right-hand side is similar.

\langle Let the hit box be the new starting box \rangle

We let the box in the hit region be the new starting box and go on to the next
intermediate plane.

  \% New start box
  xmin = ymin; xmax = ymax;
  D = dd*D;

\langle Transform the box to new coordinates with ψ \rangle

At this point we have a box on top of the cube. This box is transformed with
the coordinate change \( z = \psi(x) \) into the coordinates \( z \) where the Lorenz-Stenflo
system is a system written in normal form. We use the function xpsi_box to
determine a box in the new coordinates that contains the transformed box (this is done inside \texttt{xpsi\_box} by solving the same type of optimisation problems as in \texttt{stepwise\_box}).

\begin{verbatim}
[\texttt{zmin},\texttt{zmax}] = \texttt{xpsi\_box}(\texttt{xmin},\texttt{xmax});
[\texttt{z},\texttt{dd}] = \texttt{xpsi}(0.5*(\texttt{xmin}+\texttt{xmax}));
\texttt{D} = \texttt{dd*D};
\end{verbatim}

We also multiply $D$ by the differential of this transformation at some point inside the box. The plan is to keep on composing $D$ with the differentials of individual mappings so as in the end get an approximation of the differential of a mapping from Poincaré plane to Poincaré plane.

\begin{quote}
\textbf{\langle Let the box flow through the cube \rangle}
\end{quote}

The resulting box will now flow through the cube $\{|z_i| \leq \frac{1}{10}\}$ with the linear system and since the box is situated across the stable manifold $z_1 = 0$ of the origin it is cut into two pieces and the two halves flow on different sides of the stable manifold. On each of the two faces $z_1 = \pm \frac{1}{10}$ we use the function \texttt{linflow\_box} to calculate a box that contains the hit image of one of the half-boxes as it exits the cube. In the function \texttt{linflow\_box} we also add a small amount of excess to the enclosing box due to approximation of the normal form flow with the flow of the linear system. This excess amount is calculated in Result 7 in Chapter 3.

\begin{verbatim}
\texttt{for S = [-1 1]}
    if \texttt{S<0}, \texttt{fprintf(1,\'\n Left-hand side\n\'); end;}
    if \texttt{S>0}, \texttt{fprintf(1,\'\n Right-hand side\n\'); end;}
    \texttt{OD} = \texttt{D;}
    \texttt{[\texttt{wmin},\texttt{wmax}] = linflow\_box(\texttt{zmin},\texttt{zmax},\texttt{S},\texttt{cube\_radius},1.15e-8);}
    \texttt{z = 0.5*(\texttt{zmin}+\texttt{zmax});}
    if \texttt{S==-1}, \texttt{z(1)} = 0.5*(\texttt{zmin(1)+0}); end;
    if \texttt{S==+1}, \texttt{z(1)} = 0.5*(\texttt{0+zmax(1)}); end;
    \texttt{[\texttt{w},\texttt{dd}] = linflow(\texttt{z},\texttt{S},\texttt{cube\_radius});}
    \texttt{OD} = \texttt{dd*OD;}
\end{verbatim}

\begin{quote}
\textbf{\langle Transform back to the original coordinates with $\phi$ \rangle}
\end{quote}

Each one of the boxes on the two faces $z_1 = \pm \frac{1}{10}$ is then transformed back to the original coordinates with $x = \phi(z)$ and with the function \texttt{xphi\_box} we determine a box in coordinates $x$ that encloses the transformed box.

\begin{verbatim}
[\texttt{wmin},\texttt{wmax}] = \texttt{xphi\_box(\texttt{wmin},\texttt{wmax});}
[\texttt{w},\texttt{dd}] = \texttt{xphi(0.5*(\texttt{wmin}+\texttt{wmax});}
\end{verbatim}
CHAPTER 2. THE NUMERICAL METHOD

Figure 2.14: The box on top of the cube flows on different sides of the plane \( z_1 = 0 \) and exits the cube on the two faces \( z_1 = \pm \frac{1}{10} \).

\[
\text{OD} = \text{dd*OD};
\]

\[
\text{output\_origin\_data(S,box,wmin,wmax)};
\]

〈Find all boxes in the Poincaré plane hit by box〉

If we refer back to Figure 2.11 we now have completed steps ① to ③ and the remaining step is to let the box flow back to the Poincaré plane. This is the same situation as in step 3 and we will use the same shooting algorithm to find all boxes in the Poincaré plane that intersect the landing set of the box. We initialise the algorithm by mapping one point from the box to the Poincaré plane. The variables that store the mapping data are the same as in step03.m.

\[
\begin{align*}
\% \text{ Map a point} \\
\text{SP} &= 0.5*(\text{wmin}+\text{wmax}); \quad [\text{HP},\text{DD}] = \text{R(SP)}; \\
\text{DD} &= \text{DD}(:); \quad \text{HB} = \text{to\_box(HP)}; \\
\% \text{ Find all hit boxes} \\
[\text{HB,SP,HP,DD}] &= \text{find\_hit\_boxes(\text{wmin,\text{wmax,\text{zeros(1,d+1),0,...}}}} \\
                                    &\quad \text{HB,SP,HP,DD,0,1)}; \\
\text{DD} &= \text{append\_matrix(DD,OD)}; \\
\text{output\_box\_data(-1,S,box,HB,SP,HP,DD(1:d*d,:))};
\end{align*}
\]

〈Get a new discontinuity box〉

As soon as both half-boxes are done we start working on the next box.

\[
\begin{align*}
\text{box} &= \text{get\_unfinished\_box}; \\
\text{if size(box,2)==0, done=1; end;}
\end{align*}
\]

step06.m

Step 6 consists of two phases. The first phase is when step05.m is still running. During this time step06.m will just print out a progress report of the number of
boxes that have been completed. The second phase starts when step05.m has finished and consists of reading boxes.dat, updating variables.mat and outputting new boxes in unfinished.dat.

The program step06.m has about the same layout as step04.m.

⟨Initialisation⟩
⟨Track the progress made by step05.m⟩
⟨Read box data⟩
⟨Save box data⟩

⟨Read box data⟩

The mapping data output by step05.m is in the same format as in step 3 and 4, and we use the function read_box_data to update our variables. This time the data is added to the variables IPS and OC.

• IPS. In the variable IPS all box and mapping data of the intermediate planes are recorded. IPS is a matrix and IPS(i,1) corresponds to the discontinuity box i (i is the column index of the box in DISC) and the intermediate plane on level 1 (0 = Poincaré plane, 1 = first intermediate plane,...). IPS(i,1) has three fields .left, .middle and .right that correspond to the box in the left-hand plane, the box inside the hit region and the box in the right-hand plane, respectively. Each field has the following subfields:

  – XMIN and XMAX. The coordinate range of the box.
  – N and C. These variables are only used for the .left and .right cases and are the normal vector and the constant in the equation of the plane that contains the box.
  – CNS. This matrix corresponds to the variable CNS in variables.mat. If CNS(1,j)=k then the current box is mapped across box j in the Poincaré plane and k is the column index in SPS, HPS and DS that corresponds to this mapping (only used for .left and .right).
  – CMIN, MAX and CBS are cone variables and used in later steps.

• OC. This variable holds the same kind of data as IPS but for the two exit faces of the origin cube. OC(i) corresponds to discontinuity box i and has two fields .left and .right. The fields correspond to the left and right exit faces, respectively, and each one has the following subfields XMIN, XMAX, CNS, CMIN, CMAX and CBS with the same meaning as in IPS.

27 % Update box covering
28 fprintf(1,'Updating box covering');
29 load 'input/variables';
IPS = read_iplane_data(DISC, IPS);
OC = read_origin_data(DISC, OC);
[CH, BOXES, DISC, CNS, SPS, HPS, DS, IPS, OC] = ... 
read_box_data(BOXES, DISC, CNS, SPS, HPS, DS, IPS, OC);
added_boxes = CH;
fprintf(1, '

');

⟨Save box data⟩

If any new boxes were added to the covering in step05.m we need to go back and re-run step03.m and step04.m on them (and possibly step05.m and step06.m after that). If this is not the case we have processed all discontinuity boxes so we add them to the data file variables.mat and move on to step 7.

% Create unfinished.dat, ...

n = size(added_boxes,2);
if n~=0
    unfinished = added_boxes';
    inprogress = [];
    finished = BOXES(:,end-n+1:end)';
    save 'result/unfinished.dat' unfinished -ASCII;
    save 'result/inprogress.dat' inprogress -ASCII;
    save 'result/finished.dat' finished -ASCII;
    save 'result/variables' BOXES DISC CNS SPS HPS DS IPS OC;
    save 'result/added_boxes' added_boxes;
    fprintf(1,'%d new boxes added
',n);
    fprintf(1,'Next, do the following steps
');
    fprintf(1,' cd ../step03-04
');
    fprintf(1,' make from05-06
');
    fprintf(1,'and then start STEP03 and STEP04.

');
else
    save 'result/variables' BOXES DISC CNS SPS HPS DS IPS OC;
    fprintf(1,'No new boxes added\n',n);
    fprintf(1,'Next, do the following steps\n');
    fprintf(1,' cd ../step07
');
    fprintf(1,' make from05-06\n');
    fprintf(1,'and then start STEP07.

');
end;

2.6  Step 7

Let us summarise the results of the previous steps. We now have
• an invariant collection of boxes covering the attractor, meaning that each box in the collection is mapped with the return map inside the collection.

• that if the image of box \( i \) intersects box \( j \) then we have a point in box \( i \) that is mapped inside box \( j \). Moreover, we have the differential of the return map at that point. (This is not entirely true for the orbits that flow close to the origin. These orbits are approximated in step 5 and 6 by subdividing them into mappings from one intermediate plane to the next and the differential of the mapping from Poincaré plane and back is approximated by multiplying the differentials of these submappings.)

In this step we will start to construct inner and outer cones with the following properties.

• The two types of cones are constant on each box.

• The inner cone of a box is inside the the outer cone of that box and also inside the outer cones of neighbouring boxes.

• If box \( i \) is mapped to box \( j \) then the outer cone of box \( i \) is mapped with the differential of the return map (meaning the constant value of the differential evaluated at the point in box \( i \) that is mapped inside box \( j \)) inside the inner cone of box \( j \).

Such a set of inner and outer cones is called invariant (under this piecewise constant approximation of the differential map). The goal is to create an invariant set of cones where the cones are as small as possible.

In this step we will only create an initial set of cones (satisfying the first two properties) and in step 8 and 9 push these cones forward to the whole collection of boxes and successively enlarge the cones until we get an invariant set of cones. It is important that the cones we generate in this step are chosen sensibly since they will influence the resulting invariant cones.

The observation underlying our choice of initial cones is that near the discontinuity of the return map the differential has one large singular value and two small singular values. Because of this, almost any cone in the boxes crossing the discontinuity is mapped to a very thin cone practically coinciding with the image direction of the most expanding direction. It is therefore inevitable that any set of invariant cones must at least contain these most expanding image directions.

As our initial set of inner cones we will, consequently, pick the boxes that intersect the images of the discontinuity boxes and for each such hit box we take the most expanding directions landing in the box, and coming from the discontinuity boxes, and then choose a cone which contains these directions. This will be the initial inner cones and we go on and create on each such box an outer cone that contains the inner cone of the box and its neighbouring boxes.
step07.m

The main part of step07.m is an iteration over all boxes and on the boxes hit by any of the discontinuity boxes we create initial cones.

\[ \text{for } j = 1 : N \% \text{ Iterate over hit boxes} \]
\[ \text{endfor} \]

\[ \text{Create outer cones} \]
\[ \text{Save cone data} \]

\[ \text{Renumber the boxes of the covering} \]

When we created the box covering of the return map attractor back in step 3 to 6, boxes were added to the covering more or less haphazardly and since we will not add any new boxes this is a good time to rearrange the boxes in a more orderly fashion. We sort the boxes according to their first index. In this way it will be easier to to correlate the number of a box and its position on the attractor. It is also necessary to readjust the variables CNS, IPS and OC to correspond to the new enumeration.

\[ [\text{BOXES}, \text{I}] = \text{sortrows(BOXES');} \]
\[ \text{BOXES = BOXES';} \]
\[ \text{CNS = CNS(I,[I;I+N]);} \]
\[ \text{for } i=1:\text{size(DISC,2)} \]
\[ \text{for } l=1:\text{size(interplanes,2)} \]
\[ \text{IPS(i,l).left.CNS = IPS(i,l).left.CNS(:,[I;I+N]);} \]
\[ \text{IPS(i,l).right.CNS = IPS(i,l).right.CNS(:,[I;I+N]);} \]
\[ \text{end;} \]
\[ \text{OC(i).left.CNS = OC(i).left.CNS(:,[I;I+N]);} \]
\[ \text{OC(i).right.CNS = OC(i).right.CNS(:,[I;I+N]);} \]
\[ \text{end;} \]

\[ \text{Initialise the inner cones} \]

The inner cones we will use are constructed in the following way:

1. Pick a unit vector $u_1$ which we call the central direction of the cone.
2. Perpendicular to this direction there is a plane which we place with its origin at the tip of the vector $u_1$, and we choose an orthonormal basis $\{u_2, \ldots, u_d\}$ in this plane.

3. Select a rectangle in the plane with sides parallel to the coordinate axes. The rectangle defines a cone as the positive and negative cones spanned by its corner vectors (corner vector = a vector that starts from the base point of $u_1$ and ends in a corner point of the rectangle).

The inner cones are then stored in the following variables

- **IBS** ($d^2 \times N$-matrix). If the basis we have chosen for an inner cone is $\{u_1, u_2, \ldots, u_d\}$, where $u_1$ is the central direction of the cone and the rest of the vectors $u_2, \ldots, u_d$ are a basis in the perpendicular plane, then the column in the matrix IBS that corresponds to this inner cone contains the basis (stacked up as one column).

- **IMIN**, **IMAX** ($d \times N$-matrices). After the cone basis is fixed we can define a cone by giving bounds $[u_{\text{min}}, u_{\text{max}}]$ for the rectangle in the perpendicular plane (we have put the first coordinate of $u_{\text{min}}$ and $u_{\text{max}}$ equal to 1). The columns in IMIN and IMAX contain the vectors $u_{\text{min}}$ and $u_{\text{max}}$, respectively, for the corresponding cone.

We initialise IBS to be all zero (meaning that no inner cones nor bases are defined on any box) and the variables IMIN and IMAX also to be zero (except the first coordinate which is always 1).

\begin{verbatim}
44 IBS = zeros(d*d,N);
45 IMIN = [ones(1,N); zeros(d-1,N)];
46 IMAX = [ones(1,N); zeros(d-1,N)];
\end{verbatim}

**Find all most expanding directions hitting the box**

For each box $j$ in the covering we loop over the discontinuity boxes.
for j=1:N
  dirs = [];
  for b = DISC
    i = box_index(b,BOXES);
    Box i hits box j if the entry n = CNS(i,j) is non-zero. This entry also point to
    the column number in DS where the differential of the mapping from box i to box j
    is stored.
    With a singular value decomposition of the differential we find the most expand-
    ing direction landing in box j and store it in the variable dirs.
    n = CNS(i,j);
    if n~=0
      X = reshape(DS(:,n),d,d);
      [U,S,V] = svds(X,1);
      dirs = [dirs U];
    end;
  end;
Since box i has a mirror box $\sigma(i)$ on the other branch of the attractor we will at
the same time investigate if that box hits box j, where $\sigma$ denotes the symmetry
mapping between the branches.
Because of the way CNS is built up, only containing the information of how the
boxes from one branch are mapped, we must look if there is a mapping from box i
to box $\sigma(j)$. If this is the case then we can conclude that there is also a symmetric
mapping from box $\sigma(i)$ to box j. The differentials are related as
$$dR_{j\rightarrow\sigma(i)} = d\sigma_{j\rightarrow\sigma(j)} \circ dR_{\sigma(j)\rightarrow i} \circ d\sigma_{i\rightarrow\sigma(i)}.$$
if size(dirs,2)~=0
  [cone_min, cone_max, basis] = create_cone(dirs);
  IBS(:,j) = basis(:);
  IMIN(:,j) = cone_min;
  IMAX(:,j) = cone_max;
end;
end;

The input and output arguments of \texttt{create_cone} are

\[ [\text{CMIN}, \text{CMAX}, \text{CB}] = \text{create_cone}(D,BS,\text{OCB}); \]

- \textit{D} is a matrix with the vectors as columns.
- \textit{BS}. The vectors in \textit{D} can be grouped into consecutive blocks, where vectors from one block are spanning a cone and the whole of that cone is to be included in the created cone. The number of vectors in each block is \textit{BS} (default value is 1).
- \textit{CB} is an orthogonal matrix where the columns are the basis vectors of the created cone. The first column contains the central direction of the cone.
- \textit{CMIN} and \textit{CMAX} are two column vectors that contain the minimal and maximal coordinates of the cone, respectively.
- \textit{OCB} is an optional third input argument. If this input argument is included the columns of \textit{OCB} will be used as a cone basis instead of creating a new basis. (The output argument \textit{CB} will then be equal to \textit{OCB}.)

Most of the work done in \texttt{create_cone} is in determining the cone basis and this starts with the central direction of the cone (i.e. the first basis vector) \( u_1 \) which should be pointing in a direction similar to the vectors \( \{v_i\} \) in \textit{D}. If we assume that the vectors are unit vectors we will choose \( u_1 \) as the solution to the problem

\[
\min \sum_{i=1}^{n} (u_1 \cdot v_i)^2, \quad \text{where } \|u_1\|^2 = 1,
\]

i.e. minimizing the sum \( \sum_i \cos^2 \alpha_i \) where \( \alpha_i \) is the acute angle between \( u_1 \) and \( v_i \). This is a classical optimisation problem and its solution is to choose \( u_1 \) as the image direction corresponding to the largest singular value of the matrix whose columns are \( \{v_i\} \).

The next thing is to choose the other basis vectors \( u_2, \ldots, u_d \). We prolong the vectors \( v_i \) until they reach the plane with origin at the tip of the vector \( u_1 \) and perpendicular to \( u_1 \), thus getting the vectors \( w_i = v_i / (v_i \cdot u_1) \). If we subtract the vector \( u_1 \) from the vectors \( \{w_i\} \) we get vectors in the plane originating from the
origin of the plane to the points where the vectors $w_i$ hit the plane. We now choose
the basis vectors $u_2, \ldots, u_d$ to approximate the vectors $w_i - u_1$ well by solving the
same type of minimisation problem we did for $u_1$, which means that $u_2, \ldots, u_d$ will
be the image directions corresponding to the non-zero singular values of the matrix
with columns $\{w_i - u_1\}$.

\textbf{Create outer cones}

On the boxes that have an inner cone we also define an outer cone. This outer cone
will contain the inner cone of the box and its neighbours.

All the outer cones are created using the function \texttt{create} \_\texttt{outer} \_\texttt{cones}.

\begin{verbatim}
80    fprintf(1,'Create initial outer cones');
81
82    [OMIN,OMAX,OBS,CH] = ...
83        create_outer_cones(BOXES,IMIN,IMAX,IBS,excess);
84
85    fprintf(1,'
');
\end{verbatim}

The prototype of \texttt{create} \_\texttt{outer} \_\texttt{cones} is

\begin{verbatim}
[OMIN,OMAX,OB,OCH] = ...
    create_outer_cones(B,IMIN,IMAX,IB,XS,OOMIN,OOMAX,OOB,ICH);
\end{verbatim}

where the input arguments are

- $B$ is a matrix whose columns are the boxes of the whole collection.
- $IB$ contains the cone bases for the inner cones.
- $IMIN$ and $IMAX$ are column matrices with the minimal and maximal bounds
  of each inner cone expressed in the inner cone bases listed in $IB$.
- $XS$ is an extra amount that is added to the outer cones.
- The three optional arguments $OOMIN$, $OOMAX$ and $OOB$ specify earlier outer
  cones that are to be enlarged to correspond to the new inner cones, i.e. keep-
  ing the same bases for the outer cones and only enlarging the minimal and
  maximal bounds. If these input arguments are missing then new outer cones
  will be created from scratch.
- The last optional argument $ICH$ lists those inner cones that have changed
  since we last created outer cones.

The output arguments are

- $OCH$ specify the boxes that have a new or changed outer cone.
- $OMIN$, $OMAX$ and $OB$ are all the outer cones created to accompany the inner
  cones.
2.7. STEP 8 AND 9

(Save cone data)

The inner and outer cones are stored in the file cones.mat, and the boxes with a new outer cone are stored in changed_boxes.mat.

```matlab
% Save cone data
fprintf(1,'Save cone data');
save 'result/cones' IMIN IMAX IBS OMIN OMAX OBS;
save 'result/changed_boxes' CH;
```

The last bit of information we store is the boxes that are hit by the boxes in changed_boxes.mat. This information is stored in the variable HBS and saved in hit_boxes.mat as well as in the file unfinished.dat (starting boxes for step 8 and 9).

```matlab
% Find hit boxes
I = box_index(CH,BOXES);
[X,J] = find(CNS(I,1:N));
HBS = BOXES(:,J);
[X,J] = find(CNS(I,N+1:2*N));
HBS = [HBS BOXES(:,J)];
HBS = delete_dup(HBS);
save 'result/hit_boxes' HBS;
unfinished = HBS';
save 'result/unfinished.dat' unfinished -ASCII;
```

```matlab
fprintf(1,'Next, do the following steps
');
fprintf(1,' cd ../step08-09
');
fprintf(1,' make
');
fprintf(1,'and then start STEP08 and STEP09.

');
```

2.7 Step 8 and 9

After having created a set of initial cones in step 7 we will now use an iterative method that pushes these cones onto new boxes and modifies existing cones until we eventually get a set of invariant cones on all boxes (invariant cones means that if box $i$ is mapped to box $j$ then the outer cone on box $i$ is mapped inside the inner cone on box $j$).

Each iteration starts with some initial data stored in the following files
• **changed_boxes.mat.** Boxes that have outer cones that either changed or were created in the last iteration.

• **hit_boxes.mat.** The image boxes that the boxes in `changed_boxes.mat` are mapped inside.

• **cones.mat.** All cones generated so far.

At the start of the first iteration we get this data from `step07.m`.

For each box $j$ in `hit_boxes.mat` we loop over the boxes listed in the file `changed_boxes.mat` and check if the inner cone on box $j$ contains the image of the outer cone coming from the box in `changed_boxes.mat`. If this is not the case (or if box $j$ doesn’t yet have a cone pair) then we enlarge (or create) the inner cone on box $j$ to contain those image cones.

When all hit boxes have been checked and new inner cones have been created, or old inner cones enlarged, we go through all outer cones and change (or create) some of those to make sure they contain the inner cones of the box and its neighbouring boxes. All boxes with a modified outer cone are then included in `changed_boxes.mat` for the next iteration. The file `hit_boxes.mat` and `cones.mat` are also updated.

The work in this iterative scheme is divided between the programs `step08.m` and `step09.m`. Program `step08.m` deals with individual hit boxes. It begins with a box from `hit_boxes.mat`, creates or changes its inner cone depending on the outer cones landing in the box and then restarts with a new box from `hit_boxes.mat`. Several instances of `step08.m` can be run in parallel. The other program `step09.m` manages the iterations. While `step08.m` is running `step09.m` tracks its progress by plotting and printing partial results. When `step08.m` has finished, `step09.m` collects all data, updates the outer cones and starts the next iteration by changing the data files.

**step08.m**

The program `step08.m` has the following layout:

```
(Initialisation)
(Get a hit box)
while ∼ done
   (Load cone data)
   (Progress report)
   (Find all image cones hitting the box)
   (Update inner cone)
   (Get a new hit box)
endwhile
```
2.7. **STEP 8 AND 9**

〈Get a hit box〉

As before we use three files to keep order of the boxes being processed,

- **unfinished.dat** contains hit boxes that have not yet been processed. At the start of an iteration this file contains all hit boxes.
- **inprogress.dat** holds all hit boxes that are currently being processed.
- **finished.dat** has all completed hit boxes.

To get a box we retrieve it from the file **unfinished.dat** (and move it over to **inprogress.dat**). If the file **unfinished.dat** is empty we are either in between two iterations, in which case we wait for a new iteration to start, or the last iteration has finished and we are done. All this is taken care of by the function **get_hit_box** which either returns a box (possibly after waiting some time) or an empty matrix (when no boxes are left and the last iteration has finished).

```matlab
box = get_hit_box;
if size(box,2)==0
    done = 1;
else
    j = box_index(box,BOXES);
    done = 0;
end;
```

〈Load cone data〉

Since this program works asynchronously to **step09.m** we need to load all cone data each time we start with a new hit box (a new iteration might have started).

```matlab
while ~done
    get_lock;
    load 'cones';
    load 'changed_boxes';
    rm_lock;
```

〈Find all image cones hitting the box〉

We step through the boxes in **changed_boxes.mat** and those that are mapped to our box \( j \). Each such box \( i \) has an outer cone which is the positive and negative cones spanned by a set of corner vectors. These corner vectors are mapped with the differential map to our box and we store the resulting vectors as columns in the matrix **dirs**.
At the same time we check the mapping from the symmetric box $\sigma(i)$ to box $j$. This is done by checking the transition from box $i$ to box $\sigma(j)$ and transforming the result.

**⟨Update inner cone⟩**

To see if the image cones spanned by the vectors in $\text{dirs}$ are included in the inner cone we use $\text{create_cone}$ and find the coordinate range of a cone that contains all these cones. We then compare this cone with the inner cone and if it is not completely inside the inner cone then we create a new inner cone.

```matlab
\text{dirs} = []; k = 0; for i=1:N
  \% Box $i$ --> Box $j$
  n = CNS(i,j);
  if n~=0 & ismember(BOXES(:,i)',CH','rows')
    k = k+1;
    X = reshape(OBS(:,i),d,d);
    A = reshape(DS(:,n),d,d);
    vcts = cone_corners(OMIN(:,i),OMAX(:,i));
    dirs = [dirs A*X*vcts];
    fprintf(1,'%5d',k);
    if mod(k,10)==0, fprintf(1,'
'); end;
  end;
end;

\text{At the same time we check the mapping from the symmetric box } \sigma(i) \text{ to box } j. \text{ This is done by checking the transition from box } i \text{ to box } \sigma(j) \text{ and transforming the result.}

\text{⟨Update inner cone⟩}

% Update inner cone on box j
isConeChanged = 0;
if find(IBS(:,j)^=0)
  basis = reshape(IBS(:,j),d,d);
  [cone\_min,cone\_max] = create\_cone(dirs,pow2(d-1),basis);
  isNewConeOutside = ...  
  isempty(find(cone\_min(2:d)<IMIN(2:d,j))) | ... 
  isempty(find(cone\_max(2:d)>IMAX(2:d,j)));
  if isNewConeOutside
    vcts = cone\_corners(OMIN(:,j),OMAX(:,j));
    dirs = [dirs basis*vcts];
    [cone\_min,cone\_max,basis] = create\_cone(dirs,pow2(d-1));
    isConeChanged = 1;
  end;
else
  [cone\_min,cone\_max,basis] = create\_cone(dirs,pow2(d-1));
```
2.7. **STEP 8 AND 9**

```matlab
isConeChanged = 1;
end;
```

We output new inner cones in the file `new_cones.dat` and in `step09.m` this information is gathered and used in preparing for the next iteration.

```matlab
if isConeChanged
    output_cone_data(0,0,BOXES(:,j), ...
    cone_min-[0;repmat(excess/2,d-1,1)], ...
    cone_max+[0;repmat(excess/2,d-1,1)], basis);
end;
```

Notice that we enlarge the inner cone slightly. The reason for this is that if a cone is mapped inside another cone and the image cone fits precisely into the second cone then any modification of the first cone will, ever so slightly, modify the image cone and make it necessary to change the second cone. To prevent this propagation of changing cones we add this small excess to the inner cone.

⟨**Get a new hit box**⟩

When we are done with this box we move it from `inprogress.dat` to `finished.dat` and get a new box.

```matlab
move_box(box,’inprogress.dat’,’finished.dat’);
```

```matlab
box = get_hit_box;
if size(box,2)==0
    done = 1;
else
    j = box_index(box,BOXES);
    done = 0;
end;
```

**step09.m**

The introduction of Section 2.7 gave a general description of the program `step09.m`. The program is organised in a fairly straight-forward way.

⟨**Initialisation**⟩

```matlab
while ~done
    ⟨**Load cone data**⟩
    ⟨**Track the progress made by step08.m**⟩
    ⟨**Read new inner cones**⟩
    ⟨**Create new outer cones**⟩
    ⟨**Prepare files for the next iteration**⟩
```
endwhile
⟨Save cone data⟩

⟨Load cone data⟩

At the start of an iteration we load the cone data stored in the three files changed_boxes.mat, hit_boxes.mat and cones.mat.

```
done = 0;
iteration = 0;
while ~done
  get_lock;
  load 'cones';
  load 'changed_boxes';
  load 'hit_boxes';
  rm_lock;
  iteration = iteration+1;
```

⟨Track the progress made by step08.m⟩

To give the user some information of how step08.m is progressing we plot boxes, inner and outer cones in a three-piece plot window. In the top third we plot the

![Boxes and their inner and outer cone apertures are plotted in three diagrams. Notice that the diagrams are aligned in such a way that directly below each box are the inner and outer cone apertures plotted.](image)
boxes of the covering and indicate their status with the following colours

- red, if the box is included in the file hit_boxes.mat and, hence, is being processed in the current iteration.
- black, if the box has a cone pair defined on it.

All other boxes are omitted. The middle and bottom third show the maximal size of the inner and outer cones, respectively, and we use the same colour code as in the top third. We also print out the total number of boxes in hit_boxes.mat along with a count of how many boxes that have been processed.

⟨ **Read new inner cones** ⟩

When the current iteration step has finished step08.m will have created a number of new inner cones and output them in the file cones.dat in the format

```
i
S
a1 a2 a3
cmin1 cmin2 cmin3
cmax1 cmax2 cmax3
d1 d2 d3 d4 d5 d6 d7 d8 d9
```

These six lines means that the box \((a1, a2, a3)\) have a new inner cone with coordinate range \([cmin1, cmax1] \times [cmin2, cmax2] \times [cmin3, cmax3]\) expressed in a cone basis that has columns of the matrix

\[
\begin{pmatrix}
d1 & d4 & d7 \\
d2 & d5 & d8 \\
d3 & d6 & d9
\end{pmatrix}
\]

as basis vectors (the first column is the central direction of the cone). The first two lines are not used in this case. We read this new cone data and update the inner cones.

```matlab
% Read inner cones
get_lock;
fprintf(1,'Reading new inner cones');
[CH,IMIN,IMAX,IBS,IPS,OC] = ... read_cone_data(BOXES,DISC,IMIN,IMAX,IBS,IPS,OC);
```

⟨ **Create new outer cones** ⟩

New outer cones to correspond to the new inner cones are created with the function create_outer_cones which we used for the first time in step07.m.

```matlab
```
% Create outer cones
fprintf(1,\nCreating new outer cones');
[OMIN,OMAX,OBS,CH] = create_outer_cones(BOXES,IMIN,IMAX,...
IBS,excess,OMIN,OMAX,OBS,box_index(CH,BOXES));

〈Prepare files for the next iteration〉

We initialise the next iteration by outputting the boxes that have new or changed
outer cones in the file changed_boxes.mat and unfinished.dat, and we also up-
date the files inprogress.dat, finished.dat, hit_boxes.mat and cones.mat.

if size(CH,2)==0
    fprintf(1,\nDetermining new changed boxes');
    save 'cones' IMIN IMAX IBS OMIN OMAX OBS;
    save 'changed_boxes' CH;
    
    I = box_index(CH,BOXES);
    [X,J] = find(CNS(I,1:N));
    HBS = BOXES(:,J);
    [X,J] = find(CNS(I,N+1:2*N));
    HBS = [HBS BOXES(:,J)];
    HBS = delete_dup(HBS);
    save 'hit_boxes' HBS;
    unix([\n    'rm cones.dat &&' ... 
    'touch cones.dat']);
    unfinished = HBS';
    save 'unfinished.dat' unfinished -ASCII;
    unix('rm inprogress.dat && touch inprogress.dat');
    unix('rm finished.dat && touch finished.dat');
else
    done = 1;
end;
rm_lock;
fprintf(1,\n\n');
2.8. **STEP 10 AND 11**

After all iterations have finished we have cones that are invariant under the piecewise constant approximation of the differential map. In the next two steps these cones will be used as initial cones when we create cones that are invariant under the differential map.

We save the cones in cones.mat and initialise the next two steps with the file unfinished.dat empty and finished.dat containing a dummy box (to outsmart the routine get_hit_box and let step11.m run an initial phase before step10.m starts).

```matlab
get_lock;
unix('rm unfinished.dat && touch unfinished.dat');
unix('rm inprogress.dat && touch inprogress.dat');
unix('rm finished.dat && touch finished.dat');
rnm_lock;

save 'result/cones' IMIN IMAX IBS OMIN OMAX OBS IPS OC;

CH = BOXES;
save 'result/changed_boxes' CH;

HBS = BOXES;
save 'result/hit_boxes' HBS;

unix('touch result/unfinished.dat');
finished = zeros(1,d);
save 'result/finished.dat' finished -ASCII;

save 'result/variables' BOXES DISC CNS SPS HPS DS IPS OC;
```

2.8 **Step 10 and 11**

In these two steps we will produce the final invariant set of inner and outer cones. This means that the resulting cones have the property that if box $i$ is mapped to box $j$ then the inner cone on box $j$ contains all possible images of the outer cone on box $i$ when it is mapped with the differential of the return map (and not just the piecewise constant approximation of the differential).

The plan is to use the same algorithm as in step 8 and 9: push cones forward and enlarge cones where needed until we get the invariant cones. As starting cones we pick the result cones from step 8 and 9 which are approximately invariant and hopefully will not need too much modifications to be truly invariant.

On the surface the program step10.m might seem different from step08.m but the only place where they essentially differ is in the way cone images are determined.
CHAPTER 2. THE NUMERICAL METHOD

Below we will examine the program `step10.m` and the parts that make it different from `step08.m`. After that we also go over `step11.m` which performs the same tasks as `step09.m` but, moreover, flows cones close to the origin.

**step10.m**

The layout of `step10.m` is a copy of the layout of `step08.m`.

\[\text{Initialisation}\]
\[\text{Get a hit box}\]
\[\text{while } \sim \text{done}\]
\[\text{Load cone data}\]
\[\text{Find all image cones hitting the box}\]
\[\text{Update inner cone}\]
\[\text{Get a new hit box}\]
\[\text{endwhile}\]

Compared to `step08.m` the principal difference can be found in the code section \text{\langle Find all image cones hitting the box \rangle} and this is the only piece of the program we go over in detail.

\[\text{\langle Find all image cones hitting the box \rangle}\]

If box $i$ is mapped to box $j$ (the hit box we are currently working with) then the simplest case is if box $i$ is not a box that crosses the discontinuity of the return map. This case is handled with the function `image_cone` and we store the range of the resulting image cone (expressed in the cone basis of box $j$) as a pair of columns in `CMINS` and `CMAXS`. Notice also that in the loop below we map cones from the symmetric box $\sigma(i)$ to box $j$ (via the mapping from box $i$ to box $\sigma(j)$).

```matlab
52 53 54 55
\% The image cones landing on box
CMINS = []; CMAXS = [];
kk = 0;
for j = [jj jj+N]
57 58 59 60
    if j<=N
        [ymin,ymax] = from_box(box);
        HBS = reshape(IBS(:,j),d,d);
    else
        [ymin,ymax] = from_box(xsigma_box(box));
        HBS = dxsigma*reshape(IBS(:,j-N),d,d);
    end;
62 63 64 65
    for i=1:N
```
n = CNS(i,j);  
isDiscBox = ismember(BOXES(:,i)',DISC','rows');  
isChangedBox = ismember(BOXES(:,i)',CH','rows');  

if n~=0 & isChangedBox & ~isDiscBox  
    kk = kk+1;  
    [xmin,xmax] = from_box(BOXES(:,i));  
    xmin = [xmin; pplane];  
    xmax = [xmax; pplane];  
    SBS = reshape(OBS(:,i),d,d);  
    SBS = [SBS zeros(d,1); zeros(1,d) 1];  
    SMIN = [OMIN(:,i); 0];  
    SMAX = [OMAX(:,i); 0];  
    [CMIN,CMAX] = image_cone(xmin,xmax,zeros(1,d+1),...  
                           0,SMIN,SMAX,SBS,ymin,ymax,HBS,SPS(:,n));  
    CMINS = [CMINS CMIN];  
    CMAXS = [CMAXS CMAX];  
    fprintf(1,'%5d',kk);  
    if mod(kk,10)==0, fprintf(1,'
'); end;  
end;

The function prototype of \texttt{image_cone} is

\[ [HMIN,HMAX] = \texttt{image_cone}(XMIN,XMAX,N,C,SMIN,SMAX,...  
SBS,YMIN,YMAX,HBS,X); \]

where

- \texttt{XMIN} and \texttt{XMAX} are bounds on the starting box and the cone on that box has range \([\texttt{SMIN},\texttt{SMAX}]\) in the basis \texttt{SBS}. If the input argument \texttt{N} is not a zero

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image_cone.png}
\caption{A picture illustrating the input and output arguments of the function \texttt{image_cone}.}
\end{figure}
vector then only points in the starting box that are on the plane $\mathbf{n} \cdot \mathbf{x} = C$ are used.

- \textbf{YMIN} and \textbf{YMAX} are the bounds on the hit box where we also have the cone basis \textbf{HBS}. The output arguments \textbf{[HMIN, HMAX]} are the range of a cone that contains all images of the starting cone.

- \textbf{X} is a point in the starting box mapped inside the hit box. This point will be used as a starting point in the search for bounds on the image cone.

This function finds the image cone range \textbf{[HMIN, HMAX]} by solving the following optimisation problems

$$\min/\max \left( \frac{\text{HBS}^T \circ dR(x) \circ \text{SBS} \, v}{\text{HBS}^T \circ dR(x) \circ \text{SBS} \, v} \right)_{i,j},$$

$$\begin{align*}
x & \in [\text{XMIN}, \text{XMAX}], \\
R(x) & \in [\text{YMIN}, \text{YMAX}], \\
v & \in [\text{SMIN}, \text{SMAX}].
\end{align*}$$

These problems might, at first sight, seem complicated to solve but note that the numerator and denominator of the object function are both linear in \(v\) and, moreover, linear combinations of the components of \(dR(x)\), which are non-complicated functions (away from the discontinuity).

The second type of boxes we need to deal with are those that cross the discontinuity of the return map. The cones on these boxes are transported stepwise down to the origin cube in the same way as in step 5 and the main part of that work is done in \texttt{step11.m}. In this program we map cones back to the Poincaré plane from the boxes on the side planes of the intermediate planes and from the exit faces of the origin cube.

A first phase is to loop over all intermediate planes and map cones from the side planes to box \(j\). We use the function \texttt{image_cone} with non-zero third and fourth input argument.
2.8. **STEP 10 AND 11**

```matlab
[CMIN,CMAX] = ... 
image_cone(IPS(ii,l).left.XMIN,...
IPS(ii,l).left.XMAX, IPS(ii,l).left.N, ... 
IPS(ii,l).left.C, IPS(ii,l).left.CMIN, ...
IPS(ii,l).left.CMAX, CBS, ...
ymin,ymax,HBS,SPS(:,n));
CMINS = [CMINS CMIN];
CMAXS = [CMAXS CMAX];
end;
end;
(The right-hand plane has been excluded from this listing but is very similar to
the left-hand plane.)

The second phase is to map cones from the two exit faces of the origin cube to
box \( j \) (the right-hand side has been omitted in the listing below).

```matlab
n = OC(ii).left.CNS(1,j);
if n~=0
CBS = OC(ii).left.CBS;
CBS = reshape(CBS,d+1,d+1);
[CMIN,CMAX] = ... 
image_cone(OC(ii).left.XMIN,...
OC(ii).left.XMAX, zeros(1,d+1), ... 
0, OC(ii).left.CMIN, OC(ii).left.CMAX, ...
CBS, ymin,ymax,HBS,SPS(:,n));
CMINS = [CMINS CMIN];
CMAXS = [CMAXS CMAX];
end;
```

**step11.m**

Before **step10.m** can start its work **step11.m** needs to flow cones from the boxes
that cross the discontinuity down to the intermediate planes and the origin cube.
Thereafter **step11.m** is similar to **step09.m**.

```matlab
⟨Initialisation⟩
while ~done
⟨Load cone data⟩
for i = 1 : size(DISC,2) % Iterate over discontinuity boxes
   if ismember(DISC(:,i)',CH', 'rows')
      ⟨Initialise box and cone⟩
      for yd = interplanes % Iterate over all intermediate planes
         ⟨Determine a cone in the left-hand plane⟩
      end;
   end;
end;
```

```matlab
end;
```
CHAPTER 2. THE NUMERICAL METHOD

(Determine a cone in the right-hand plane)
(Determine a cone in the next plane)
(Let the cone in the next plane be the new cone)

endfor

‘Transform the cone to new coordinates with $\psi$’

for $S = [-1 1]$ % Flow on both sides of the origin
(Let the cone flow through the origin cube)
(Transform the cone back to original coordinates with $\phi$)

endfor

endif

endfor

(Prepare step10.m for a new iteration)
(Track the progress made by step10.m)
(Read new inner cones)
(Create new outer cones)
(Prepare files for the next iteration)

endwhile

(Save cone data)

(Initialise box and cone)

As we follow a box and its cone in its descent towards the origin cube the following variables are used to keep track of the box and cone:

- $[\text{xmin}, \text{xmax}]$ is the coordinate range of the box.
- $[\text{CMIN}, \text{CMAX}]$ is the range of the cone expressed in the basis CBS.

\[
\text{xmin} = \text{from_box}([\text{DISC}(:,i)]);
\text{xmax} = [\text{xmax}; \text{pplane}];
\text{CMIN} = [\text{OMIN}(:,ii);0]; \text{CMAX} = [\text{OMAX}(:,ii);0];
\text{CBS} = [\text{reshape}([\text{OBS}(:,ii),d,d] \text{zeros}(d,1)); ... \text{zeros}(1,d) 1]);
\]

(Determine a cone in the left-hand plane)

When a box in an intermediate plane is mapped to the next plane a part of the box escapes to the left of the hit region in the next plane and lands in a left-hand plane (see Figure 2.10 in Section 2.5). In this code block we use the function stepwise_plane_cone to find a cone in the left-hand plane that contains all images of the cone in the intermediate plane after it is mapped to this plane.

% Left plane cone
if isfield(IPS(i,1).left,’XMIN’)


2.8. **STEP 10 AND 11**

```matlab
ymin = IPS(i,1).left.XMIN;
ymax = IPS(i,1).left.XMAX;
n = IPS(i,1).left.N;
C = IPS(i,1).left.C;
x = [xmin(1); 0.5*(xmin(2:end)+xmax(2:end))];
BMIN = [min([xmin(1:end-1) ymin(1:end-1)]-0.05,...
[]), 2); ymin(end)];
BMAX = [max([xmax(1:end-1) ymax(1:end-1)]+0.05,...
[]), 2); xmax(end)+0.05];
[ht, y, dy] = box_R(x, BMIN, BMAX, n, C);
[HBS, whatever] = qr(dy*CBS);
[HMIN, HMAX] = stepwise_plane_cone(xmin, xmax, ...
CMIN, CMAX, CBS, ymin, ymax, BMIN, BMAX, n, C, CBS, x);
output_cone_data(l, -1, DISC(:, i), HMIN, HMAX, BMIN);
end;
```

The cone in the right-hand plane is found in the same way.

**Determine a cone in the next plane**

To find a cone in the next intermediate plane that contains all image cones under the mapping between the two planes we use the function `stepwise_cone`.

```matlab
% Next plane
ymin = IPS(i, 1).middle.XMIN;
ymax = IPS(i, 1).middle.XMAX;
[x, y, dy] = stepwise_intersection(xmin, xmax, ymin, ymax);
[HBS, whatever] = qr(dy*CBS);
[HMIN, HMAX] = stepwise_cone(xmin, xmax, ...
CMIN, CMAX, CBS, ymin, ymax, CBS, x, y);
DH = max([HMIN(2:end-1)-HHMIN(2:end-1) ... 
repmat(1e-9, size(HMIN, 1)-2, 1)], [], 2);
HMIN = [1; 0.5*(HHMAX(2:end-1)+HHMIN(2:end-1))-0.5*DH; 0];
HMAX = [1; 0.5*(HHMAX(2:end-1)+HHMIN(2:end-1))+0.5*DH; 0];
output_cone_data(1, 0, DISC(:, i), HMIN, HMAX, CBS);
```

The function `stepwise_cone` is analogous to `image_cone` and it has the following prototype

```matlab
[HMIN, HMAX] = stepwise_cone(XMIN, XMAX, SMIN, SMAX, SBS, YMIN, YMAX, CBS, X, Y);
```

where

- **XMIN(i) ≤ xi ≤ XMAX(i)** is the starting box and **[SMIN, SMAX]** is the starting cone expressed in the basis **SBS**.
• \( Y_{\text{MIN}}(i) \leq x_i \leq Y_{\text{MAX}}(i) \) is the hit region in the next intermediate plane and \( H_{\text{BS}} \) is the cone basis used for the hit cone \([H_{\text{MIN}},H_{\text{MAX}}]\).

• \( X \) is a point in the starting box being mapped to a point \( Y \) in the hit region.

Inside \texttt{stepwise\_cone} the same kind of optimisation problems are solved as in the function \texttt{image\_cone}.

\(<\text{Let the cone in the next plane be the new cone}\>)

The box and the cone in the next intermediate plane will be a new starting box and a new starting cone for the next iteration.

\begin{verbatim}
108    xmin = ymin; xmax = ymax;
109    CMIN = HMIN; CMAX = HMAX; CBS = HBS;
\end{verbatim}

\(<\text{Transform the cone to new coordinates with } \psi\>)

When the cone has reached the top of the cube it is transformed with \( \psi \) to the coordinate system where the Lorenz-Stenflo system is in normal form. This is done with the function \texttt{xpsi\_cone} which determines a cone that contains all image cones after the transformation. It should be remarked that the cone output by \texttt{xpsi\_cone} is a cone in the full phase space (a four-dimensional cone) and is constructed in the same way as lower dimensional cones (one central direction and a rectangle in the perpendicular plane).

\begin{verbatim}
120    \% Transform with xpsi
121    HBS = eye(d+1);
122    [CMIN,CMAX] = xpsi\_cone(xmin,xmax,CMIN,CMAX,CBS,HBS);
123    [zmin,zmax] = xpsi\_box(xmin,xmax);
124    CBS = HBS;
\end{verbatim}

\(<\text{Let the cone flow through the origin cube}\>)

Next, cones flow through the cube with the linear system and exit the cube on either of the two faces \( z_1 = \pm \frac{1}{10} \) (depending on the sign of \( S \)). The function \texttt{linflow\_cone} returns a cone in the tangent space of the exit face that contains all image cones under this mapping by solving the following problems

\[
\min / \max \left( \begin{array}{c}
H_{\text{BS}}^T \circ d\tilde{L}(z) \circ S_{\text{BS}} v \\
H_{\text{BS}}^T \circ d\tilde{L}(z) \circ S_{\text{BS}} v
\end{array} \right)_i,
\]

under the appropriate constraints. In this formula \( \tilde{L}(z) \) denotes the mapping from the top of the cube onto either of the exit faces when points flow according to the
linear system. Since we are using the linear system as opposed to the normal form system the cone apertures we obtain should be widened by the amount

\[
\frac{\left| (HBS^T \circ dL(z) \circ SBS v)_i \right|}{\left| (HBS^T \circ dL(z) \circ SBS v)_1 \right|} - \frac{\left| (HBS^T \circ d\tilde{L}(z) \circ SBS v)_i \right|}{\left| (HBS^T \circ d\tilde{L}(z) \circ SBS v)_1 \right|}
\]

to compensate for this approximation. The function \( L(z) \) corresponds to \( \tilde{L}(z) \) but flows orbits with the normal form system instead. By using the elementary inequality

\[
\left| \frac{y + \Delta y}{x + \Delta x} - \frac{y}{x} \right| \leq \max\left\{ \frac{\Delta x, |\Delta y|}{|x| - |\Delta x|} \right\} \left( 1 + \frac{|y|}{|x|} \right),
\]

where \(|x| > |\Delta x|\), the amount we need to add to cones is less than

\[
\frac{\left\| HBS^T \circ (dL(z) - d\tilde{L}(z)) \circ SBS v \right\|_\infty}{\left| HBS(:,1)^T \circ d\tilde{L}(z) \circ SBS v \right|} - \frac{\left| HBS(:,1)^T \circ (dL(z) - d\tilde{L}(z)) \circ SBS v \right|}{\left| HBS(:,1)^T \circ d\tilde{L}(z) \circ SBS v \right|} \times \left( 1 + \max\left| \frac{HBS(:,1)^T \circ d\tilde{L}(z) \circ SBS v}{HBS(:,1)^T \circ dL(z) \circ SBS v} \right| \right).
\]

In Section 3.5 of Chapter 3 we derive the estimate \( \|dL(z) - d\tilde{L}(z)\|_\infty \leq 6.52 \times 10^{-7} \) \((z_1/r)^{-2-M\lambda_4/\lambda_1}\) and if we use this on the last formula we can replace the upper bound on the cone aperture change with

\[
6.52 \times 10^{-7} \left\| HBS^T \right\|_\infty \left\| SBS \right\|_\infty - 6.52 \times 10^{-7} \left\| HBS^T \right\|_\infty \left\| SBS \right\|_\infty \times \left( 1 + \max\left| \frac{HBS(:,1)^T \circ d\tilde{L}(z) \circ SBS v}{HBS(:,1)^T \circ dL(z) \circ SBS v} \right| \right).
\]

The maximisation problem in this formula is something we already need to solve when calculating the cone aperture so the only additional problem we have is

\[
\min \left| \frac{HBS(:,1)^T \circ d\tilde{L}(z) \circ SBS v}{(z_1/r)^{-2-M\lambda_4/\lambda_1}} - \frac{6.52 \times 10^{-7} \left\| HBS^T \right\|_\infty \left\| SBS \right\|_\infty}{\left( z_1/r \right)^{-2-M\lambda_4/\lambda_1}} \right| \times \left( 1 + \max\left| \frac{HBS(:,1)^T \circ d\tilde{L}(z) \circ SBS v}{HBS(:,1)^T \circ dL(z) \circ SBS v} \right| \right).
\]

and since \( d\tilde{L}(z) \) can be calculated explicitly this problem can easily be solved. One thing to observe when solving this problem is that the minimal value is positive only if \( \lambda_1 + (M - 1)\lambda_4 < 0 \) (an assumption we make in Chapter 3 and which reappears here).

Special attention must also be paid to the points that start with \( z_1 = 0 \). These points are on the stable manifold and will converge to the origin. For these points the routine \texttt{linflow_cone} does the proper thing and returns the limiting behaviour of cones as they tend to \( z_1 = 0 \).
% Flow through cube
for S=[-1 1]
    HBS = [0 0 0 1; 0 1 0 0; 0 0 1 0]’;
    [HMIN,HMAX] = linflow_cone(zmin,zmax,CMIN,CMAX,CBS,...
        HBS,S,cube_radius,6.52e-7);
    [wmin,wmax] = linflow_box(zmin,zmax,S,cube_radius,...
        1.15e-8);

⟨Transform the cone back to the original coordinates with φ⟩

We then transform the cones back to the ordinary coordinates with xphi_cones which works in the same way as xpsi_cones.

% Transform with xphi
SBS = [HBS [1;0;0;0]];
SMIN = [HMIN;0]; SMAX = [HMAX;0];
[x,dx] = xphi(0.5*(wmin+wmax));
[HBS,whatever] = qr(dx*SBS);
[HMIN,HMAX] = xphi_cone(wmin,wmax,SMIN,SMAX,SBS,HBS);
output_cone_data(-1,S,DISC(:,i),HMIN,HMAX,HBS);
end;

⟨Prepare step10.m for a new iteration⟩

Once all cones on the discontinuity boxes have been transported down to the origin we update the variables IPS and OC, remove the dummy box in finished.dat and add the boxes from hit_boxes.mat to unfinished.dat.

get_lock;
[CH,IMIN,IMAX,IBS,IPS,OC] = ... 
    read_cone_data(BOXES,DISC,IMIN,IMAX,IBS,IPS,OC);
    save ’cones’ IMIN IMAX IBS IPS OC;
    unix(’rm finished.dat && touch finished.dat’);
    load ’hit_boxes’;
    unfinished = HBS(:,size(HBS,2):-1:1)’;
    save ’unfinished.dat’ unfinished -ASCII;
rm_lock;

The rest of the program is similar to step09.m.

2.9 Step 12 and 13

If we look at the work done so far it has resulted in a box covering of the return map attractor and on each box a pair of invariant cones: inner and outer cones. The outer cone on a box contains the inner cone of the same box and also the inner
cones of neighbouring boxes. The invariance means that the outer cone is mapped with the differential of the return map inside the inner cone of the box it is mapped to.

In these two steps we will extend the previous results and for each mapping from one box to another box calculate the smallest expansion factor a vector inside the outer cone of the first box is subjected to when it is mapped to the second box.

With this information we can build a graph of the box covering and its mapping data. The nodes in this graph are the boxes and a directed edge from box $i$ to box $j$ with weight $w$ means that box $i$ is mapped to box $j$ and the logarithm of the smallest expansion factor of this mapping is $w$. This graph will be analysed in step 14 and 15 to see if we can conclude that the system possesses a strange attractor.

The work is divided between the programs `step12.m` and `step13.m`. The program `step12.m` picks a box from the covering and looks at all boxes that are mapped to this box. For each such mapping it calculates the smallest logarithmic expansion factor and outputs it in the file `logexpansion.dat`. When all mappings to the box have been processed the program starts over with a new box. This continues until no box remains. The accompanying program `step13.m` is run in parallel with `step12.m`. First it calculates the log-expansion of the mappings from discontinuity boxes down to the intermediate planes and the origin cube. (The last mapping, back to the Poincaré plane, is done in `step12.m`.) After this is done `step13.m` tracks how `step12.m` is progressing and when `step12.m` has finished it collects the output from `step12.m` and prepares for the next two steps.

**step12.m**

The program `step12.m` is organised in the following way:

```plaintext
  ⟨Initialisation⟩
  ⟨Get a hit box⟩
  while ~done
    ⟨Calculate the smallest log-expansion of cones hitting the box⟩
    ⟨Get a new hit box⟩
  endwhile

Most of these code sections are standard and it is only in the code section where the logarithmic expansion is calculated that the actual computations are performed.

 ⟨Get a hit box⟩

We have the usual system of three box files `unfinished.dat`, `inprogress.dat` and `finished.dat` to keep track of boxes and the state they are in.

```python
23  box = get_hit_box;
24  if size(box,2)==0
25    done = 1;
```
else
    jj = box_index(box, BOXES);
    done = 0;
end;

(Calculate the smallest log-expansion of cones hitting the box)

As we iterate through the boxes in the covering we divide the boxes that are mapped to box \( j \) into two classes: boxes that cross the discontinuity and those that do not. The mappings from the second type of boxes are easiest to deal with and the logarithmic expansion is calculated with the function \texttt{image_logexpansion}.

% The expansion inside the cone
kk = 0;
for j = [jj jj+N]
    if \( j < N \)
        [ymin, ymax] = from_box(box);
    else
        [ymin, ymax] = from_box(xsigma_box(box));
    end;
    for i=1:N
        LES = [];
        n = CNS(i, j);
        isDiscBox = ismember(BOXES(:, i)', DISC', 'rows');
        if n==0 & ~isDiscBox
            kk = kk+1;
            [xmin, xmax] = from_box(BOXES(:, i));
            xmin = [xmin; pplane];
            xmax = [xmax; pplane];
            SBS = reshape(OBS(:, i), d, d);
            SBS = [SBS zeros(d,1); zeros(1,d) 1];
            SMIN = [OMIN(:, i); 0];
            SMAX = [OMAX(:, i); 0];
            LE = image_logexpansion(xmin, xmax, zeros(1, d+1), 0,...
                SMIN, SMAX, SBS, ymin, ymax, SPS(:, n));
            LES = [LES LE];
            fprintf(1, '%5d', kk);
            if mod(kk, 10)==0, fprintf(1, '
'); end;
        end;
    end;
The prototype of this function is

$$\text{LE} = \text{image\_logexpansion}(\text{XMIN}, \text{XMAX}, N, C, \text{SMIN}, \text{SMAX}, \text{SBS}, \text{YMIN}, \text{YMAX}, X);$$

where $[\text{XMIN}, \text{XMAX}]$ is a starting box and $N \cdot x = C$ is a plane in the box that the starting points are restricted to (if $N = 0$ and $C = 0$ then this plane vanishes). The output LE is the smallest logarithmic expansion inside the cone $[\text{SMIN}, \text{SMAX}]$ expressed in the basis SBS when points from the starting box are mapped to the box $[\text{YMIN}, \text{YMAX}]$.

What this function does is to solve the problem

$$\min \log \frac{\|dR(x) \circ \text{SBS} v\|_2}{\|v\|_2}$$

$$\begin{cases} 
  x \in [\text{XMIN}, \text{XMAX}] \\
  N \cdot x = C \\
  R(x) \in [\text{YMIN}, \text{YMAX}] \\
  v \in [\text{SMIN}, \text{SMAX}] 
\end{cases}$$

and this might seem straightforward, but notice that, disregarding the monotone logarithmic function, the object function behaves typically like a quadratic function and it is possible for it to have a local minimum on the boundary of the cone which is not a global minimum. This phenomena can be illustrated with a two-dimensional cone having edge vectors $v_1$ and $v_2$, and which is mapped according to Figure 2.18. In this case $v_1$ is a local minimum (nearby vectors inside the cone are expanded more than $v_1$) but the global minimum is in $v_2$. The example can be complicated further when $dR$ varies with $x$ and the global minimum can flip over from $v_2$ to $v_1$, and vice versa, in different parts of the domain. To avoid that

![Figure 2.18: This figure illustrates how vectors expand when mapped with the differential map. The expansion factor can have local minima on the boundary that are not a global minimum.](image)

the numerical optimisation routine returns a non-global minimum of the expansion the function image\_logexpansion solves the optimisation problem for each lower-dimensional subface of the cone boundary.
When it comes to the first type of mappings (from boxes that cross the discontinuity) these are split up, as before, into several parts: mapping stepwise down to the cube around the origin, through the cube and back to the Poincaré plane. In this program only the last part is performed (*step13.m* takes care of the rest). There are two types of mappings involved: mapping from the sides of the intermediate planes and from the exit faces of the origin cube. Both these kind of mappings are calculated with *image_logexpansion*. (The right-hand side calculations have been omitted in the listing below.)

```matlab
if n~=0 & isDiscBox
    kk = kk+1;
    ii = box_index(BOXES(:,i),DISC);
    for l=1:size(interplanes,2)
        if isfield(IPS(ii,l).left,'XMIN')
            n = IPS(ii,l).left.CNS(1,j);
            if n~=0
                CBS = IPS(ii,l).left.CBS;
                CBS = reshape(CBS,d+1,d+1);
                LE = image_logexpansion(IPS(ii,l).left.XMIN, ...
                    IPS(ii,l).left.XMAX, IPS(ii,l).left.N, ...
                    IPS(ii,l).left.C, IPS(ii,l).left.CMIN, ...
                    IPS(ii,l).left.CMAX, CBS, ...
                    ymin, ymax, SPS(:,n));
                LES = [LES IPS(ii,l).left.LE+LE];
            end;
        end;
    end;
    n = OC(ii).left.CNS(1,j);
    if n~=0
        CBS = OC(ii).left.CBS;
        CBS = reshape(CBS,d+1,d+1);
        LE = image_logexpansion(OC(ii).left.XMIN, ...
            OC(ii).left.XMAX, zeros(1,d+1), ...
            0, OC(ii).left.CMIN, OC(ii).left.CMAX, ...
            CBS, ymin, ymax, SPS(:,n));
        LES = [LES OC(ii).left.LE+LE];
    end;
    fprintf(1,'%5d',kk);
    if mod(kk,10)==0, fprintf(1,'
'), end;
end;
```

80   CHAPTER 2. THE NUMERICAL METHOD
2.9. **STEP 12 AND 13**

```matlab
138     end;
139
140     if size(LES,2)~=0
141         LE = min(LES,[],2);
142         output_logexpansion(i,j,LE);
143     end;
144
145     end;
146     end;
```

**step13.m**

The program `step13.m` is organised in a similar way to `step11.m`. First it calculates the logarithmic expansion of the boxes that flow down to the intermediate planes and the origin cube. Then the work is mostly administrative. It monitors the progress `step12.m` is making and rearranges the data output by `step12.m` in a suitable format.

```matlab
⟨Initialisation⟩
for i = 1 : size(DISC,2)  % Iterate over discontinuity boxes
    ⟨Initialise box and cone⟩
    for yd = interplanes  % Iterate over all intermediate planes
        ⟨Calculate log-expansion of the mapping to the left-hand plane⟩
        ⟨Calculate log-expansion of the mapping to the right-hand plane⟩
        ⟨Calculate log-expansion of the mapping to the next plane⟩
        ⟨Let the cone in the next plane be the new cone⟩
    endfor
    ⟨Calculate log-expansion of transformation with $\psi$⟩
    ⟨Calculate log-expansion of the mapping through the left half cube⟩
    ⟨Calculate log-expansion of the mapping through the right half cube⟩
endfor
⟨Prepare data files for step12.m⟩
⟨Track the progress made by step12.m⟩
⟨Read expansion data⟩
⟨Prepare files for the next steps⟩
```

⟨Initialise box and cone⟩

The same variables as in `step11.m` are used to keep track of the current box and cone.

```matlab
29     [xmin,xmax] = from_box(DISC(:,i));
30     xmin = [xmin; pplane]; xmax = [xmax; pplane];
```
CHAPTER 2. THE NUMERICAL METHOD

31 \[ \text{CMIN} = [\text{DMIN}(;ii);0]; \quad \text{CMAX} = [\text{OMAX}(;ii);0]; \]
32 \[ \text{CBS} = [\text{reshape}([\text{OBS}(;ii),d,d); \text{zeros}(d,1); \text{zeros}(1,d); 1]; \]

Calculate log-expansion of the mapping to the left-hand plane

To go from one intermediate plane to the next we first calculate the logarithmic expansion of the mapping to the left-hand plane of the next intermediate plane with the function \text{stepwise\_plane\_logexpansion}. This function solves the same type of optimisation problem as in \text{image\_logexpansion}.

42 \%
43 \text{Left plane cone}
44 \text{if isfield(IPS(i,l).left,'XMIN')}
45 \quad \text{ymin} = \text{IPS(i,l).left.XMIN};
46 \quad \text{ymax} = \text{IPS(i,l).left.XMAX};
47 \quad \text{n} = \text{IPS(i,l).left.N};
48 \quad \text{C} = \text{IPS(i,l).left.C};
49 \quad \text{x} = [\text{xmin}(1); 0.5*(\text{xmin}(2:end)+\text{xmax}(2:end))];
50 \quad \text{BMIN} = [\text{min}([\text{xmin}(1:end-1) \text{ymin}(1:end-1)];...}
51 \quad \text{\text{ymin}(end)];}
52 \quad \text{BMAX} = [\text{max}([\text{xmax}(1:end-1) \text{ymax}(1:end-1)];+0.05,[,2]);...}
53 \quad \text{\text{xmax}(end)+0.05];}
54 \quad \text{LE} = \text{stepwise\_plane\_logexpansion(xmin,xmax,CMIN,CMAX,CBS,...}
55 \quad \quad \text{ymin,ymax,BMIN,BMAX,n,C,x);}
56 \text{if l>1}
57 \quad \text{IPS(i,l).left.LE = IPS(i,l-1).middle.LE+LE};
58 \text{else}
59 \quad \text{IPS(i,l).left.LE = LE};
60 \text{end;}

The mapping to the right-hand side is handled in the same way.

Calculate log-expansion of the mapping to the next plane

We find the logarithmic expansion of the mapping to the next intermediate plane by using the function \text{stepwise\_logexpansion} which is similar to the functions \text{image\_logexpansion} and \text{stepwise\_plane\_logexpansion}.

82 \%
83 \text{Next plane}
84 \text{ymin} = \text{IPS(i,l).middle.XMIN};
85 \text{ymax} = \text{IPS(i,l).middle.XMAX};
86 \text{[x,y,dy] = stepwise\_intersection(xmin,xmax,ymin,ymax);}
87 \text{LE} = \text{stepwise\_logexpansion(xmin,xmax,CMIN,CMAX,CBS,...}
88 \quad \quad \text{ymin,ymax,x,y);}
89 \text{if l>1}
2.9. **STEP 12 AND 13**

\[ IPS(i,l).middle.LE = IPS(i,l-1).middle.LE + LE; \]
\[ \text{else} \]
\[ IPS(i,l).middle.LE = LE; \]
\[ \text{end;} \]

〈**Let the cone in the next plane be the new cone**〉

The new box and cone become the starting box and starting cone of the next iteration.

\[ \text{xmin} = \text{ymin}; \text{xmax} = \text{ymax}; \]
\[ \text{CMIN} = IPS(i,l).middle.CMIN; \]
\[ \text{CMAX} = IPS(i,l).middle.CMAX; \]
\[ \text{CBS} = \text{reshape}(IPS(i,l).middle.CBS,d+1,d+1); \]

〈**Calculate log-expansion of transformation with } \psi \text{**〉

Cones flow through the origin cube via the usual coordinate changes and linear flow. We use the function \text{xpsi_logexpansion} to calculate the logarithmic expansion of the first coordinate change.

\[ \% \text{ Transform with xpsi} \]
\[ \text{LE} = \text{xpsi_logexpansion}(\text{xmin},\text{xmax},\text{CMIN},\text{CMAX},\text{CBS}); \]
\[ \text{LE} = \text{LE} + IPS(i,l).middle.LE; \]
\[ \text{HBS} = \text{eye}(d+1); \]
\[ [\text{CMIN},\text{CMAX}] = \text{xpsi_cone}(\text{xmin},\text{xmax},\text{CMIN},\text{CMAX},\text{CBS},\text{HBS}); \]
\[ [\text{zmin},\text{zmax}] = \text{xpsi_box}(\text{xmin},\text{xmax}); \]
\[ \text{CBS} = \text{HBS}; \]

〈**Calculate log-expansion of the mapping through the left half cube**〉

When cones flow through the cube we approximate the normal form system with the linear system and solve the problem

\[ \min \log \frac{\|d\tilde{L}(z) \circ \text{SBS} v\|_2}{\|v\|_2}, \]

where \( \tilde{L}(z) \) is the map through the cube that uses the linear system to integrate orbits as opposed to the map \( L(z) \) that uses the normal form system. This approximation introduces an error in the expansion calculation and to correct this we use
that the object function in the normal form case \((dL\) instead of \(d\tilde{L}\)) is greater than

\[
\log \frac{\|d\tilde{L}(z) \circ SBS \, v\|_2}{\|v\|_2} - \frac{\|(dL(z) - d\tilde{L}(z)) \circ SBS \, v\|_2}{\|v\|_2}
\]

\[
\geq \log \frac{\|d\tilde{L}(z) \circ SBS \, v\|_2}{\|v\|_2} + \log \left( 1 - \sqrt{3} \frac{\|dL(z) - d\tilde{L}(z)\|_\infty \|SBS\|_\infty}{\|d\tilde{L}(z) \circ SBS \, v\|_2} \right)
\]

\[
\geq \log \frac{\|d\tilde{L}(z) \circ SBS \, v\|_2}{\|v\|_2} + \log \left( 1 - \sqrt{3} \frac{6.52 \times 10^{-7} \cdot \max \|SBS \, v\|_\infty}{\min \frac{\|d\tilde{L}(z) \circ SBS \, v\|_2}{\|v\|_2}} \right)
\]

where we also have used Result 9 in Chapter 3. The calculations done in the function \texttt{linflow\_logexpansion} use this formula to subtract the effect of using the “wrong” system in the optimisation.

```matlab
118 \% Flow through the left half of the cube
119 LE = linflow_logexpansion(zmin,zmax,CMIN,CMAX,CBS,-1,...
120 cube_radius,6.52e-7);
121 LE = OLE + LE;
122 HBS = [0 0 0 1; 0 1 0 0; 0 0 1 0]';
123 [HMIN,HMAX] = linflow_cone(zmin,zmax,CMIN,CMAX,CBS,...
124 HBS,-1,cube_radius,6.52e-7);
125 [wmin,wmax] = linflow_box(zmin,zmax,-1,cube_radius,1.15e-8);

The smallest expansion of the second coordinate change \(\phi\) is calculated with the function \texttt{xphi\_logexpansion} which is analogous to \texttt{xpsi\_logexpansion}.

```matlab
126 SBS = [HBS [1;0;0;0]]; SMIN = [HMIN;0]; SMAX = [HMAX;0];
127 LE = LE + xphi_logexpansion(wmin,wmax,SMIN,SMAX,SBS);
128 OC(i).left.LE = LE;
```

Flowing through the right-hand side of the cube works the same way.

\textbf{(Read expansion data)}

When \texttt{step12.m} has finished all expansion data is stored in a three-line format in the output file \texttt{logexpansion.dat}

\[
i
\]
\[
j
\]
\[
le
\]

which means that box \(i\) is mapped to box \(j\) and the smallest logarithmic expansion of that mapping is \(le\). This information is saved in the sparse matrix \(LE\) as \(LE(i,j) = le\), and we then include this variable in the file \texttt{variables.mat}.  


% Read expansion data
fprintf(1,'Reading expansion data');
load 'variables';
N = size(BOXES,2);
[I,J] = find(CNS);
LE = sparse(I,J,repmat(Inf,size(I,1),1),N,2*N);
LE = read_logexpansion(LE);

(Prepare files for the next steps)
The variable LE will actually not be used by later steps but is used more for debugging purposes. The real output is made to the file boxgraph_main.dat in a row oriented format

\begin{verbatim}
a i j le
\end{verbatim}

where "a" is a literal letter while i, j and le means as before that box i is mapped to box j with logarithmic expansion le. This is a standard format for encoding graphs and in the next steps we will use two graph analysing programs to investigate the shortest cycles and paths in the graph of boxes.

% Create boxgraph_main.dat
fprintf(1,'
Creating boxgraph_main.dat');
[fptr,errmsg] = fopen('result/boxgraph_main.dat','w');
if fptr==-1, error(errmsg); end;
for i=1:N
    for j=1:2*N
        n = CNS(i,j);
        if n~=0
            fprintf(fptr,'a %d %d %12.12f
',i,j,LE(i,j));
            if j<=N
                fprintf(fptr,'a %d %d %12.12f
',i+N,j+N,LE(i,j));
            else
                fprintf(fptr,'a %d %d %12.12f
',i+N,j-N,LE(i,j));
            end;
        end;
    end;
end;
fclose(fptr);

2.10 Step 14 and 15

All mapping data created in the previous steps is put together in a graph. In this graph the nodes are the boxes and an edge from box i to box j corresponds to the fact that box i is mapped to box j. Each edge has a weight equal to the smallest logarithmic expansion factor of that mapping. If this graph doesn’t contain any
cycles with negative total weight then this translates into the fact that all orbits
of the return map and the induced map have positive Lyapunov exponent, which
is a crucial part in showing that there exists a strange attractor. Another part is
showing that all paths in the graph between two boxes that cross the discontinuity
of the return map must have total weight greater than \( \log 2 \). This property implies
that the attractor is topologically transitive (see page 12).

Seen from a graph theory perspective we have two standard problems

- show that a graph doesn’t contain any negative cycles, and
- find the shortest (least weight) path between two nodes.

In the programs \texttt{step14.m} and \texttt{step15.m} we solve these problems and the results
can be found in Chapter 4.

\textbf{step14.m}

We use a C-program \texttt{bfct} to determine if the box graph has a negative cycle or not.
This program is part of a bundle \texttt{spc} of several similar programs written by Boris
V. Cherkassky and Andrew V. Goldberg in connection with a study of algorithms
to solve this problem [\textit{Mathematical Programming} \textbf{85} (1999), pages 277–311]. It
can be downloaded from the web page

\texttt{http://www.avglab.com/andrew/soft.html}

The program \texttt{step14.m} adds a few lines to the graph file \texttt{boxgraph_main.dat} and starts \texttt{bfct}. The output of this program is redirected to the file \texttt{neg_cycles.dat}
in the subdirectory \texttt{result/} for later inspection.

```matlab
15 clear;
16 load 'input/variables';
17 path_to_bfct = 'spc-1.2/bin/';
18
19 N = size(BOXES,2);
20 M = size(find(CNS),1);
21
22 [fptr,errmsg] = fopen('boxgraph.dat','w');
23 if fptr==-1, error(errmsg); end;
24 fprintf(fptr,'t Box graph\n');
25 fprintf(fptr,'p sp %d %d\n',2*N,2*M);
26 fprintf(fptr,'n 1\n');
27 fclose(fptr);
28
29 unix('cat input/boxgraph_main.dat >> boxgraph.dat');
30 unix([path_to_bfct 'bfct <boxgraph.dat ' ...
31     'result/neg_cycles.dat']);
```
step15.m

The second graph problem is solved with a program gor1 which is part of the bundle splib that Boris V. Cherkassky, Andrew V. Goldberg and Tomasz Radzik wrote when they studied algorithms for determining shortest paths in a graph [Mathematical Programming 73 (1996), pages 129–174]. It can be downloaded from the same web page as spc.

By using gor1 we find the shortest paths from one selected node to the rest of the nodes in the graph. When we do this for all nodes in the graph we get all shortest paths and then it is simple to determine all paths between the discontinuity boxes and print them.

An outline of program step15.m is:

⟨Initialisation⟩

for i = 1 : 2 * N
  ⟨Find all shortest paths from node i⟩
  ⟨Store shortest paths data⟩
endfor

⟨Add shortest cycles data⟩

⟨Output result⟩

⟨Initialisation⟩

We store the information of the shortest paths in the two matrices

• C, where the element \((i,j)\) is the weight of the shortest path from node \(i\) to node \(j\).

• Theta, where the element \((i,j)\) is the second to last node in the shortest path from node \(i\) to node \(j\) (the last node being \(j\)).

These variables are appropriately initialised.

```matlab
clear;
p = pwd; cd('..'); parent=pwd; cd(p);
path(path,[parent '/m-files']);
load 'input/variables';
path_to_gor1 = 'splib-1.4/bin/';
tmpdir = './';
N = size(BOXES,2);
M = size(find(CNS),1);
C = repmat(Inf,2*N,2*N);
Theta = repmat([1:2*N]',1,2*N);
```
CHAPTER 2. THE NUMERICAL METHOD

〈Find all shortest paths from node i〉

The program gor1 is fed the graph data through the file boxgraph.dat which we create by prepending a few lines to the file boxgraph_main.dat created in step13.m. The output is redirected to the file short_paths.dat.

```matlab
fprintf(1,'Find shortest paths\n');
fprintf(1,' Total %d\n',N);
for i=1:N
    [fptr,errmsg] = fopen('boxgraph.dat','w');
    if fptr==-1, error(errmsg); end;
    fprintf(fptr,'t Box graph\n');
    fprintf(fptr,'p sp %d %d\n',2*N,2*M);
    fprintf(fptr,'n %d\n',i);
    fclose(fptr);
    unix('cat input/boxgraph_main.dat >> boxgraph.dat');
    [s,m] = unix(['rm ' tmpdir 'short_paths.dat']);
    [s,m] = unix([path_to_gor1 'gor1 <boxgraph.dat | tail +10 ' ...
                   '>' tmpdir 'short_paths.dat']);
```

〈Store shortest paths data〉

Each line in the output file short_paths.dat has the format

\[
i \ j \ k \ w
\]

which means that the shortest path from node i to node j has weight w and that the second to last node in this path is k.

```matlab
[fptr,errmsg] = fopen(['tmpdir ' short_paths.dat'],'r');
if fptr==-1, error(errmsg); end;
while 1
    line = fgetl(fptr);
    if ~ischar(line), break, end;
    tmp = sscanf(line,'%f');
    ii = round(tmp(1));
    jj = round(tmp(2));
    kk = round(tmp(3));
    C(ii,jj) = tmp(4);
    Theta(ii,jj) = kk;
end;
fclose(fptr);
```
Add shortest cycles data

To find the shortest cycle that contains node $i$ (which is not explicitly calculated by gor1) we will for each other node $j$ in the graph find the shortest path from $i$ to $j$ and then back from $j$ to $i$. In this way we will find the shortest cycle going via node $j$. We can then compare all these cycles and determine the overall shortest cycle. The result is stored in the diagonal elements of the matrices $C$ and $\Theta$.

```
\% Fill in the diagonal elements of $C$ and $\Theta$
for i=1:2*N
    C(i,i) = Inf;
    for j=1:2*N
        if j~i
            if C(i,j)+C(j,i)<C(i,i)
                C(i,i) = C(i,j)+C(j,i);
                Theta(i,i) = Theta(j,i);
            end;
        end;
    end;
end;
```

Output result

In the result file we print the overall smallest expansion of a cycle and the shortest paths between each pair of discontinuity boxes.
Chapter 3

The coordinate change around the origin

Arguably the most troublesome area in analysing the Lorenz or Lorenz-Stenflo system is the origin. The origin is a fixed point of saddle type and this means that there are orbits, i.e. those that flow close to the stable manifold, that stay an arbitrary long time in a neighbourhood of the origin. Our standard numerical integrator routines are unable to approximate these orbits well due to the long integration times involved but also because these orbits form a very small subset of all orbits and this makes them difficult to home in on.

What follows in this chapter is an account of a method that uses so-called normal forms and were first introduced in this context by Warwick Tucker to produce rigorous results in connection with this particular problem. Our way of presenting the normal form method is in essence the same as Warwick Tucker’s although we differ in some details and we opt to work with recurrence equations directly instead of the corresponding generating functions. We will only do a thorough derivation for the Lorenz-Stenflo system and refer to the last section of this chapter for the corresponding results for the Lorenz system.

3.1 The normal form method

The Lorenz-Stenflo system can be written as

\[
\frac{d}{dt} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\
0 & \lambda_2 & 0 & 0 \\
0 & 0 & \lambda_3 & 0 \\
0 & 0 & 0 & \lambda_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + F(x)
\]

or in an abbreviated form \( \dot{x} = Ax + F(x) \), where \( F(x) \) represents the second order terms; see formula (1) in Chapter 2. Around the origin, inside a cube of side length \( \frac{1}{5} \), we will use an analytic change of coordinates \( x = \phi(z) \), and its
inverse \( z = \psi(x) \), that transforms the Lorenz-Stenflo system to an approximately linear system

\[
\frac{d}{dt} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} + G(z),
\]

where \( G(z) \) satisfies \( |G_i(z)| \leq C |z_1|^M (|z_2| + |z_3| + |z_4|)^M \), which means that \( G \) is very small close to the stable and unstable manifolds.

This is an example of a normal form transformation that pushes the non-linear terms in \( F \) to higher order terms and makes the system closer to a linear system. As we shall see, the existence of this kind of transformation depends on whether there are resonances between the diagonal elements \( \lambda_1, \lambda_2, \lambda_3 \) and \( \lambda_4 \).

The idea is that we will use these coordinates when orbits enter the cube. In the new coordinates we can flow points through the cube by approximating the normal form system with the linear system (see Section 3.5) and thereby avoid any numerical difficulties caused by long integration times by using closed-form formulas. When orbits exit the cube we change back to the original coordinates and continue with the standard integrator routine. A more detailed description of this procedure can be found in Section 2.5 of Chapter 2.

In the upcoming sections we will first go over how the coordinate change functions \( \phi \) and \( \psi \) are calculated and estimate the error in the approximations involved. Then we take a look at the non-linear term \( G(z) \) in the normal form system and derive estimates of it as well as of its derivative. In the section thereafter we show that the normal form flow, inside the cube mentioned above, can be well approximated by the flow of the linear system. In the final section we give the corresponding results for the Lorenz system.

### 3.2 The coordinate change function \( \phi \)

Our algorithm for calculating \( \phi \) involves several steps. First we will derive a functional equation for \( \phi \) and in order to solve this equation we write \( \phi \) as a (formal) power series (using multi-index notation)

\[
\phi_i(z) = \sum_j c_{i,j} z^j = \sum_{j_1,j_2,j_3,j_4} c_{i,j_1,j_2,j_3,j_4} z_1^{j_1} z_2^{j_2} z_3^{j_3} z_4^{j_4}.
\]

When this series is substituted into the functional equation we get recurrence equations for the coefficients \( c_{i,j} \). Using the recurrence equations we calculate all coefficients up to degree \( N \), i.e. coefficients \( c_{i,j} \) for \( |j| = j_1 + j_2 + j_3 + j_4 \leq N \), and approximate \( \phi \) by truncating its power series

\[
\phi_i(z) = \sum_{|j| \leq N} c_{i,j} z^j + \text{Remainder term}.
\]
Next we start working on estimating the remainder term in the approximation. The remainder term is split in two parts

\[ R_1 + R_2 = \sum_{N < |j| \leq N'} c_{i,j} z^j + \sum_{|j| > N'} c_{i,j} z^j. \]

For the first part we introduce quantities \( h_{i,j} \) which are greater than the sums of the absolute values of the \( c_{i,j} \) for fixed values of \( |j| \),

\[ h_{i,j} \geq \sum_{|j|=j} |c_{i,j}|, \]

and satisfy a simpler recurrence equation than that for the coefficients \( c_{i,j} \). Inside the cube \( |z_i| \leq \frac{1}{10} \) we can then estimate the first part with

\[ |R_1| \leq \sum_{j=N+1}^{N'} h_{i,j} \left( \frac{1}{10} \right)^j. \]

For the second part we prove an inequality of the form \( h_{i,j} \leq C_i K^j \), for \( j > N' \), and can estimate the remaining series with

\[ |R_2| \leq C_i \sum_{j=N'+1}^{\infty} \left( \frac{K}{10} \right)^j. \]

Similar estimates are also made for the derivatives of \( \phi \).

The above calculations furthermore show that the power series of \( \phi \) converges inside the domain \( |z_i| < 1/K \), but to assure that the whole coordinate transformation is analytic we need to show that the power series of the non-linear function \( G \) is also convergent and this is done in Section 3.4.

**Recurrence equations**

There are two different ways of expressing the derivative \( \dot{x} \) in terms of the new coordinate \( z \),

\[ \dot{x} = Ax + F(x) = A\phi(z) + F(\phi(z)) \]

and

\[ \dot{x} = \frac{d}{dt} x = \frac{d}{dt} \phi(z) = D\phi(z) \dot{z} = D\phi(z)(Az + G(z)). \]

Putting these two expressions equal shows that \( \phi \) must satisfy the following functional equation

\[ A\phi(z) + F(\phi(z)) = D\phi(z)(Az + G(z)) \]

\[ \Leftrightarrow \quad D\phi(z)Az - A\phi(z) = F(\phi(z)) - D\phi(z)G(z). \quad (*) \]
CHAPTER 3. THE COORDINATE CHANGE AROUND THE ORIGIN

To solve this equation for $\phi$ we make the following Ansatz for $\phi$ and $G$ as power series

$$\phi_i(z) = \sum_j c_{i,j} z^j$$ and $$G_i(z) = \sum_j d_{i,j} z^j,$$

and identify coefficients on both sides of the equation. Starting with the left-hand side of (*) we have that

$$\left(D\phi(z)A z - A\phi(z)\right)_i = \sum_{k=1}^4 \left(\frac{\partial}{\partial z_k} \sum_j c_{i,j} z^j\right) \lambda_k z_k - \sum_j \lambda_i c_{i,j} z^j$$

$$= \sum_j \left(\sum_{k=1}^4 \lambda_k j_k - \lambda_i\right) c_{i,j} z^j.$$

It is worth noting that the monomials $\phi(z) = z^j$ are all eigenfunctions of the linear operator in the left-hand side and this fact simplifies the recurrence equations that will follow considerably.

The right-hand side is more complex than the left-hand side so we treat the two terms separately. The first term expands to

- $F_1(\phi(z)) = k_1(\phi_1(z) + \phi_2(z) + \phi_3(z))\phi_4(z)$
  
  $$= k_1 \left(\sum_m (c_{1,m} + c_{2,m} + c_{3,m}) z^m\right) \left(\sum_n c_{4,n} z^n\right)$$
  
  $$= \sum_j k_1 \left(\sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}\right) z^j,$$

- $F_2(\phi(z)) = \sum_j k_2 \left(\sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}\right) z^j,$

- $F_3(\phi(z)) = \sum_j k_3 \left(\sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}\right) z^j,$

- $F_4(\phi(z)) = \sum_j \left(\sum_m (k_{4,1,m} c_{1,j-m} + k_{5,1,m} c_{2,j-m} + k_{6,1,m} c_{3,j-m} + k_{7,1,m} c_{2,j-m} + k_{8,1,m} c_{3,j-m} + k_{9,1,m} c_{3,j-m})\right) z^j,$

and using the notation $e_k$ for the $k$th unit vector the second term is equal to

$$\left(D\phi(z)G(z)\right)_i = \sum_{k=1}^4 \frac{\partial}{\partial z_k} \phi_i(z) \cdot G_k(z)$$

$$= \sum_{k=1}^4 \left(\sum_m c_{i,m} m_k z^{m-e_k}\right) \left(\sum_n d_{k,n} z^n\right)$$
3.2. THE COORDINATE CHANGE FUNCTION $\phi$

\[
= \sum_j \sum_m \left( c_{i,m+e_1}(m_1 + 1)d_{1,j-m} \right.
+ c_{i,m+e_2}(m_2 + 1)d_{2,j-m}
+ c_{i,m+e_3}(m_3 + 1)d_{3,j-m} \left. + c_{i,m+e_4}(m_4 + 1)d_{4,j-m} \right) z^j.
\]

When we identify both sides of (*) we end up with the following relations between the coefficients

- $\Omega_1(j) c_{1,j} = k_1 \sum_m (c_{1,m} + c_{2,m} + c_{3,m})c_{4,j-m}$
  + $\sum_m (c_{1,m+e_1}(m_1 + 1)d_{1,j-m} + c_{1,m+e_2}(m_2 + 1)d_{2,j-m}$
  + $c_{1,m+e_3}(m_3 + 1)d_{3,j-m} + c_{1,m+e_4}(m_4 + 1)d_{4,j-m})$, 

- $\Omega_2(j) c_{2,j} = k_2 \sum_m (c_{1,m} + c_{2,m} + c_{3,m})c_{4,j-m}$
  + $\sum_m (c_{2,m+e_1}(m_1 + 1)d_{1,j-m} + c_{2,m+e_2}(m_2 + 1)d_{2,j-m}$
  + $c_{2,m+e_3}(m_3 + 1)d_{3,j-m} + c_{2,m+e_4}(m_4 + 1)d_{4,j-m})$, 

- $\Omega_3(j) c_{3,j} = k_3 \sum_m (c_{1,m} + c_{2,m} + c_{3,m})c_{4,j-m}$
  + $\sum_m (c_{3,m+e_1}(m_1 + 1)d_{1,j-m} + c_{3,m+e_2}(m_2 + 1)d_{2,j-m}$
  + $c_{3,m+e_3}(m_3 + 1)d_{3,j-m} + c_{3,m+e_4}(m_4 + 1)d_{4,j-m})$, 

- $\Omega_4(j) c_{4,j} = \sum_m \left( k_4 c_{1,m}c_{1,j-m} + k_5 c_{1,m}c_{2,j-m} + k_6 c_{1,m}c_{3,j-m} \right.$
  + $k_7 c_{2,m}c_{2,j-m} + k_8 c_{2,m}c_{3,j-m} + k_9 c_{3,m}c_{3,j-m} \left. \right)$
  + $\sum_m \left( c_{4,m+e_1}(m_1 + 1)d_{1,j-m} + c_{4,m+e_2}(m_2 + 1)d_{2,j-m} \right.$
  + $c_{4,m+e_3}(m_3 + 1)d_{3,j-m} + c_{4,m+e_4}(m_4 + 1)d_{4,j-m} \right)$,

where the short-hand notation $\Omega_i(j) = j_1 \lambda_1 + j_2 \lambda_2 + j_3 \lambda_3 + j_4 \lambda_4 - \lambda_i$ is an integer linear combination of the elements $\lambda_1, \lambda_2, \lambda_3$ and $\lambda_4$. It is easy to see that since $c_{i,0} = 0$ and $d_{i,j} = 0$ when $|j| \leq 1$, we must have that $c_{i,j} = 0$ when $|j| = 1$ but $j \neq e_i$. If $j = e_i$ then $\Omega_i(j) = 0$ and we can choose $c_{i,e_i}$ freely, for example $c_{i,e_i} = 1$ so that $\phi$ approximates the identity map. Other than this the formulas above only express a relationship between the $c$- and $d$-coefficients and do not determine them uniquely. This freedom will be used to ensure that $G$ only contains higher order terms, i.e. that $d_{i,j} = 0$ when $|j|$ is small, and thus making the
system approximately linear. The precise formulation we choose is that $G$ should satisfy
\[ |G_i(z)| \leq C |z_1|^M (|z_2| + |z_3| + |z_4|)^M \]
in a neighbourhood of the origin, which translates into that $d_{i,j} = 0$ if $j_1 \leq M - 1$ or $j_2 + j_3 + j_4 \leq M - 1$. This means that $G$ is especially small close to the stable manifold $z_1 = 0$ and the unstable manifold $z_2 = z_3 = z_4 = 0$, which is an important fact when we approximate the normal form flow with the linear flow (orbits that stay a long time inside the cube spend most of that time close to either of these two manifolds).

A consequence of this is that when $j_1 \leq M - 1$ or $j_2 + j_3 + j_4 \leq M - 1$ we have that $d_{i,j-m} = 0$ for all $m \geq 0$ and the recurrence relations reduce to

1. $\Omega_1(j) c_{1,j} = k_1 \sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}$, \hspace{1cm} (1)
2. $\Omega_2(j) c_{2,j} = k_2 \sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}$, \hspace{1cm} (2)
3. $\Omega_3(j) c_{3,j} = k_3 \sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}$, \hspace{1cm} (3)
4. $\Omega_4(j) c_{4,j} = \sum_m \left( k_4 c_{1,j-m} c_{1,j-m} + k_5 c_{1,m} c_{2,j-m} + k_6 c_{1,m} c_{3,j-m} \\
+ k_7 c_{2,m} c_{2,j-m} + k_8 c_{2,m} c_{3,j-m} + k_9 c_{3,m} c_{3,j-m} \right)$. \hspace{1cm} (4)

These relations express $c_{i,j}$, for $j$ in this parameter range, in terms of lower order coefficients $c_{i,m}$ (where $|m| < |j|$) and we can therefore use them to gradually determine values for the coefficients $c_{i,j}$ unless $\Omega_i(j)$ happen to be zero. If the linear combination $\Omega_i(j)$ degenerates to zero for some $j$ we have what is called a resonance between the lambda elements and this is known in the normal form theory to be the obstacle that prevents one from eliminating higher order terms via an analytic coordinate change. This is also the reason why we do not try to choose all $d_{i,j} = 0$, i.e. $G \equiv 0$, because even if $\Omega_i(j) \neq 0$ for all $j$ the values of $\Omega_i(j)$ will get arbitrarily close to 0 and this leads to a more complicated analysis as mentioned in Section 1.4 (remember that we will anyhow be able to approximate the orbits of $\dot{z} = A\dot{z} + G(\dot{z})$ with the corresponding orbits of $\dot{z} = A\dot{z}$ in Section 3.5). We would on top of that also lose any form of robustness when it comes to small changes in the parameters $k_1, \ldots, k_9$.

In the case when $j_1 \geq M$ and $j_2 + j_3 + j_4 \geq M$ we can rewrite the recurrence relations on page 95 in the form
\[ \Omega_i(j) c_{i,j} = \sum_m \left( \cdots \right) + \sum_{|m| \geq 1} \left( \cdots \right) + d_{i,j} \]
and observe that $c_{i,j}$ and $d_{i,j}$ are simultaneously expressed in terms of lower order coefficients. We can use this relation to avoid the problem with degeneration
3.2. THE COORDINATE CHANGE FUNCTION $\phi$

of $\Omega_i(j)$ by letting $c_{i,j} = 0$ and compensate by choosing $d_{i,j}$ accordingly,

- $d_{1,j} = -k_1 \sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}$
  \[ - \sum_{|m| \geq 1} (c_{1,m+e_1}(m_1 + 1)d_{1,j-m} + c_{1,m+e_2}(m_2 + 1)d_{2,j-m} + c_{1,m+e_3}(m_3 + 1)d_{3,j-m} + c_{1,m+e_4}(m_4 + 1)d_{4,j-m}), \]  
  \( (5) \)

- $d_{2,j} = -k_2 \sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}$
  \[ - \sum_{|m| \geq 1} (c_{2,m+e_1}(m_1 + 1)d_{1,j-m} + c_{2,m+e_2}(m_2 + 1)d_{2,j-m} + c_{2,m+e_3}(m_3 + 1)d_{3,j-m} + c_{2,m+e_4}(m_4 + 1)d_{4,j-m}), \]  
  \( (6) \)

- $d_{3,j} = -k_3 \sum_m (c_{1,m} + c_{2,m} + c_{3,m}) c_{4,j-m}$
  \[ - \sum_{|m| \geq 1} (c_{3,m+e_1}(m_1 + 1)d_{1,j-m} + c_{3,m+e_2}(m_2 + 1)d_{2,j-m} + c_{3,m+e_3}(m_3 + 1)d_{3,j-m} + c_{3,m+e_4}(m_4 + 1)d_{4,j-m}), \]  
  \( (7) \)

- $d_{4,j} = -\sum_m (k_4 c_{1,m} c_{1,j-m} + k_5 c_{1,m} c_{2,j-m} + k_6 c_{1,m} c_{3,j-m} + k_7 c_{2,m} c_{2,j-m} + k_8 c_{2,m} c_{3,j-m} + k_9 c_{3,m} c_{3,j-m})$
  \[ - \sum_{|m| \geq 1} (c_{4,m+e_1}(m_1 + 1)d_{1,j-m} + c_{4,m+e_2}(m_2 + 1)d_{2,j-m} + c_{4,m+e_3}(m_3 + 1)d_{3,j-m} + c_{4,m+e_4}(m_4 + 1)d_{4,j-m}). \]  
  \( (8) \)

Let us summarise. We saw at the beginning of this section that the coordinate change function $\phi$ satisfies a functional equation (⋆) that also involves the new right-hand side function $G$, and by writing $\phi$ and $G$ as power series this led to recurrence relations between the coefficients $c_{i,j}$ of $\phi$ and the coefficients $d_{i,j}$ of $G$. These recurrence relations do not determine the coefficients completely since the function $G$ is only specified by certain constraints. We use this to choose certain coefficients $c_{i,j}$ to be equal to zero and at the same time simplify the recurrence equations so that

- when $j_1 \leq M - 1$ or $j_2 + j_3 + j_4 \leq M - 1$ the coefficients $c_{i,j}$ are defined by the recurrence equations (1) to (4), and $d_{i,j} = 0$.

- when $j_1 \geq M$ and $j_2 + j_3 + j_4 \geq M$ all coefficients $c_{i,j}$ are zero and the coefficients $d_{i,j}$ are given by the recurrence equations (5) to (8).
The first part of the remainder term

Next on the agenda is to study the first part of the remainder term of \( \phi \),

\[
R_1 = \sum_{N < |j| \leq N'} c_{i,j} z^j,
\]

inside the cube \(|z_i| \leq \frac{1}{10}\). As mentioned earlier we estimate this part by introducing a sequence \( \{h_{i,j}\}_{j=N+1}^{N'} \) of numbers that majorize the sums

\[
h_{i,j} \geq \sum_{|j|=j} |c_{i,j}|,
\]

because then we have that

\[
|R_1| \leq \sum_{j=N+1}^{N'} (\max_k |z_k|)^j \sum_{|j|=j} |c_{i,j}| \leq \sum_{j=N+1}^{N'} h_{i,j}(\frac{1}{10})^j.
\]

For the first few values of \( h_{i,j} \) we use the recurrence equations of \( c_{i,j} \) and calculate \( h_{i,j} \) as

\[
h_{i,j} = \sum_{|j|=j} |c_{i,j}| \quad \text{for } j = N+1, \ldots, P,
\]

but as \( j \) increases this becomes less and less feasible. Instead, for larger \( j \) we derive a simpler, yet cruder, recurrence equation for \( h_{i,j} \). This recurrence equation results from taking absolute value on both sides of the recurrence equations (1)–(4) for \( c_{i,j} \) and summing over the parameter set \( J_j = \{j \geq 0 : |j| = j \text{ and } (j_1 < M \text{ or } j_2 + j_3 + j_4 < M)\} \). Starting with \( c_{1,j} \), i.e. equation (1), we get that

\[
\text{LHS} = \sum_{j \in J_j} |\Omega_1(j)||c_{1,j}| \geq \Omega_1(j) \sum_{j \in J_j} |c_{1,j}| = \Omega_1(j) \sum_{|j|=j} |c_{1,j}|,
\]

\[
\text{RHS} \leq |k_1| \sum_{j \in J_j} \sum_{m=1}^{j-1} \sum_{|m|=m} |c_{1,m} + c_{2,m} + c_{3,m}| |c_{4,j-m}|
\]

\[
\leq |k_1| \sum_{j \in J_j} \sum_{m=1}^{j-1} \sum_{|m|=m} |c_{1,m} + c_{2,m} + c_{3,m}| |c_{4,j-m}|
\]

\[
= |k_1| \sum_{m=1}^{j-1} \sum_{|m|=m} |c_{1,m} + c_{2,m} + c_{3,m}| \sum_{|j|=j} |c_{4,j-m}|
\]

\[
\leq |k_1| \sum_{m=1}^{j-1} h_{4,j-m} \sum_{|m|=m} (|c_{1,m}| + |c_{2,m}| + |c_{3,m}|)
\]

\[
\leq |k_1| \sum_{m=1}^{j-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m}),
\]
where $\Omega_i(j) = \min_{j \in J, \Omega_i(j)} |\Omega_i(j)|$. The result is that we can use the following recurrence equation to define the sequence $h_{i,j}$:

$$h_{1,j} = \frac{|k_1|}{\Omega_1(j)} \sum_{m=1}^{j-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m}) \quad \text{for } j > P.$$ 

Since the recurrence equations for $c_{2,j}$ and $c_{3,j}$ are very similar to the recurrence equation for $c_{1,j}$ we can put

$$h_{2,j} = \frac{|k_2|}{k_1} \frac{\Omega_1(j)}{\Omega_2(j)} h_{1,j} \quad \text{and} \quad h_{3,j} = \frac{|k_3|}{k_1} \frac{\Omega_1(j)}{\Omega_3(j)} h_{1,j}$$

for $j \geq 2$. When we do the corresponding calculations for $h_{4,j}$ we arrive at

$$h_{4,j} = \frac{1}{\Omega_4(j)} \sum_{m=1}^{j-1} \left( |k_4|h_{1,m}h_{1,j-m} + |k_5|h_{1,m}h_{2,j-m} + |k_6|h_{1,m}h_{3,j-m} + |k_7|h_{2,m}h_{2,j-m} + |k_8|h_{2,m}h_{3,j-m} + |k_9|h_{3,m}h_{3,j-m} \right).$$

This sequence $\{h_{i,j}\}$ that we have just defined plays an important role in our estimate of the remainder term, not only in this first part of the remainder term but also in the second part. It is therefore necessary to study and understand this sequence better.

Probably the most important factor that controls the size of $h_{i,j}$ is the integer linear combinations in $\Omega_i(j)$. If the elements $\lambda_1$, $\lambda_2$, $\lambda_3$ and $\lambda_4$ happen to be close to a resonance, i.e. $\Omega_i(j)$ very close to zero for a certain $j$, this could blow up the value of $h_{i,j}$ and possibly make our estimate useless.

![Figure 3.1](image.png)

**Figure 3.1**: A combined diagram plotting the values of $\Omega_1(j)$, $\Omega_2(j)$, $\Omega_3(j)$ and $\Omega_4(j)$. Note that both axes have a logarithmic scale.

In Figure 3.1 there is a plot of $\Omega_i(j)$ and we can observe that the minimal value of any $\Omega_i(j)$ is $\Omega_3(16) \approx 3 \times 10^{-3}$. This might seem worrying but we must keep in mind that this is a one-time occurrence and, moreover, since the recurrence
equations for \( h_{1,j}, h_{2,j}, h_{3,j} \) and \( h_{4,j} \) are coupled this makes the four sequences interrelated which tends to smooth out big differences.

Another observation we make is that when \( j \) is larger than a certain value the quantities \( \Omega_i(j) \) will grow linearly. The explanation of this phenomena is that in the minimisation

\[
\Omega_i(j) = \min_{j \in J_j} |j_1 \lambda_1 + j_2 \lambda_2 + j_3 \lambda_3 + j_4 \lambda_4 - \lambda_i|
\]

the parameter set \( J_j = \{ j \geq 0 : |j| = j \text{ and } (j_1 < M \text{ or } j_2 + j_3 + j_4 < M) \} \) is such that for large enough \( j \) one must have either that \( j_1 \leq M - 1 \) or \( j_2 + j_3 + j_4 \leq M - 1 \), but not both, and this means that one term in the linear combination will dominate. After a tedious calculation one can in fact show that (assuming that \( M \geq 3 \))

- \( \Omega_1(j) = j|\lambda_4|-(M-1)(|\lambda_1|+|\lambda_4|)+|\lambda_1| \) when \( j \geq (M-1)\frac{|\lambda_1|+|\lambda_4|}{|\lambda_4|} - \frac{|\lambda_1|}{|\lambda_4|} \)
- \( \Omega_2(j) = j|\lambda_4|-(M-1)(|\lambda_1|+|\lambda_4|)-|\lambda_2| \) when \( j \geq (M-1)\frac{|\lambda_1|+|\lambda_4|}{|\lambda_4|} + \frac{|\lambda_2|}{|\lambda_4|} \)
- \( \Omega_3(j) = j|\lambda_4|-(M-1)(|\lambda_1|+|\lambda_4|)-|\lambda_3| \) when \( j \geq (M-1)\frac{|\lambda_1|+|\lambda_4|}{|\lambda_4|} + \frac{|\lambda_3|}{|\lambda_4|} \)
- \( \Omega_4(j) = j|\lambda_4|-(M-1)(|\lambda_1|+|\lambda_4|)-|\lambda_4| \) when \( j \geq (M-1)\frac{|\lambda_1|+|\lambda_4|}{|\lambda_4|} + 1 \)

In Table 3.1 we have listed values of \( h_{i,j} \) around \( j = P \) and notice the distinct turn at \( j = P + 1 \) where we start using the recurrence equation to calculate \( h_{i,j} \). Instead of slowly decreasing the sequence starts to grow rapidly. Evidently, approximating with the recurrence equation gives a very rough upper bound on the sums \( \sum_{|j|=j} |c_{i,j}| \), but as long as the sequence \( \{h_{i,j}\} \) doesn’t grow as fast as \( 10^j \) we can get some form of estimate of the remainder term.

<table>
<thead>
<tr>
<th>j</th>
<th>( h_{1,j} )</th>
<th>( h_{2,j} )</th>
<th>( h_{3,j} )</th>
<th>( h_{4,j} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2.49×10⁻³</td>
<td>1.35×10⁻⁴</td>
<td>1.22×10⁻³</td>
<td>2.92×10⁻³</td>
</tr>
<tr>
<td>8</td>
<td>3.82×10⁻⁴</td>
<td>3.04×10⁻⁵</td>
<td>2.32×10⁻⁴</td>
<td>2.02×10⁻³</td>
</tr>
<tr>
<td>9</td>
<td>3.13×10⁻⁴</td>
<td>1.43×10⁻⁵</td>
<td>1.09×10⁻⁴</td>
<td>3.99×10⁻⁴</td>
</tr>
<tr>
<td>10</td>
<td>5.03×10⁻³</td>
<td>8.64×10⁻⁴</td>
<td>5.85×10⁻²</td>
<td>2.57×10⁻²</td>
</tr>
<tr>
<td>11</td>
<td>6.49×10⁻₂</td>
<td>1.11×10⁻²</td>
<td>1.07×10⁰</td>
<td>7.65×10⁻¹</td>
</tr>
<tr>
<td>12</td>
<td>2.74×10⁰</td>
<td>4.70×10⁻¹</td>
<td>8.86×10⁰</td>
<td>2.39×10¹</td>
</tr>
</tbody>
</table>

**Table 3.1:** A tabulation of \( h_{i,j} \) around the threshold value \( j = P = 9 \).

The Maple program **phi_error.mpl** implements this estimation scheme and by choosing the various parameters as \( M = 6 \), \( N = 8 \), \( P = 9 \) and \( N' = 40 \) the result is that \( |R_1| \) is smaller than \( 1.75×10⁻⁹ \). See Table 3.2 on page 103.
3.2. THE COORDINATE CHANGE FUNCTION $\phi$

The second part of the remainder term

By extending Table 3.1 we can extrapolate that the sequence $\{h_{i,j}\}$ grows something like $\sim C_i K^j$, where $K \approx 3.7$, and using induction we will prove that

$$h_{i,j} \leq C_i K^j \quad \text{for} \quad j > N',$$

where we have chosen $K = 5$ (somewhat larger than the optimal value in order to make the induction work). When we have this estimate the last part of the remainder term can be bounded by

$$|R_2| \leq \sum_{j=N'+1}^{\infty} h_{i,j} \left( \frac{1}{10} \right)^j \leq C_i \sum_{j=N'+1}^{\infty} \left( \frac{K}{10} \right)^j.$$

The induction is fairly straightforward. The induction hypothesis is that

$$h_{i,m} \leq C_i K^m \quad \text{for} \quad N'' \leq m \leq j - 1.$$

To start the induction we choose $C_i$ so that (*) is satisfied for $N'' \leq m \leq N' - 1$. For convenience we will also assume that $2N'' \leq N' - 1$ in order to simplify the details when we work on the induction step.

We start with $h_{1,j}$ and assuming that (*) holds we want to show that

$$h_{1,j} = \frac{|k_1|}{\Omega_1(j)} \sum_{m=1}^{j-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m}) \leq C_1 K^j.$$

We break up this sum in three parts and use the induction hypothesis (this is where it is convenient to assume that $2N'' \leq N' - 1$ so that none of the sums degenerates)

$$h_{1,j} = \frac{|k_1|}{\Omega_1(j)} \sum_{m=1}^{N''-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m})$$

$$\leq \frac{|k_1|}{\Omega_1(j)} \sum_{m=1}^{N''-1} C_4 K^{j-m} \cdot (h_{1,m} + h_{2,m} + h_{3,m})$$

$$+ \sum_{m=N''}^{j-N''} C_4 K^{j-m} \cdot (C_1 + C_2 + C_3) K^m$$

$$+ \sum_{m=j-N''+1}^{j-1} h_{4,j-m} \cdot (C_1 + C_2 + C_3) K^m$$

$$= \frac{|k_1|}{\Omega_1(j)} \left( \sum_{m=1}^{N''-1} [(h_{1,m} + h_{2,m} + h_{3,m}) C_4 + (C_1 + C_2 + C_3) h_{4,m}] K^{-m} \right.$$

$$+ \sum_{m=N''}^{j-N''} (C_1 + C_2 + C_3) C_4 \left. \right) K^j.$$
The induction step will work if

\[
\frac{|k_1|}{\Omega_1(j)} \left( \sum_{m=1}^{N''-1} \left[ (h_{1,m} + h_{2,m} + h_{3,m})C_4 + (C_1 + C_2 + C_3)h_{4,m} \right] K^{-m} \right.
\]

\[
\left. + (C_1 + C_2 + C_3)C_4 (j - 2N'' + 1) \right) \leq C_1
\]

and since

\[
j - 2N'' + 1 = \frac{j - 2N'' + 1}{\Omega_1(j)} \leq \frac{1}{|\lambda_4|}
\]

these conditions can all be replaced with the condition

\[
\frac{|k_1|}{\Omega_1(N' + 1)} \sum_{m=1}^{N''-1} \left[ (h_{1,m} + h_{2,m} + h_{3,m})C_4 + (C_1 + C_2 + C_3)h_{4,m} \right] K^{-m}
\]

\[
+ |k_1| (C_1 + C_2 + C_3)C_4 \cdot \frac{1}{|\lambda_4|} \leq C_1.
\]  \hspace{1cm} (9)

Due to the close relationship between \(h_{1,j}, h_{2,j}\) and \(h_{3,j}\) it is easy to derive conditions that yield the corresponding inequalities for \(h_{2,j}\) and \(h_{3,j}\). We have that

- \(h_{2,j} = \frac{|k_2|}{|k_1|} \frac{\Omega_1(j)}{\Omega_2(j)} h_{1,j} \leq \frac{|k_2|}{|k_1|} \frac{\Omega_1(j)}{\Omega_2(j)} C_1 K^j;\)
- \(h_{3,j} = \frac{|k_3|}{|k_1|} \frac{\Omega_1(j)}{\Omega_3(j)} h_{1,j} \leq \frac{|k_3|}{|k_1|} \frac{\Omega_1(j)}{\Omega_3(j)} C_1 K^j;\)

which means that the inequalities are satisfied if

\[
C_2 \geq \frac{|k_2|}{|k_1|} \frac{\Omega_1(N' + 1)}{\Omega_2(N' + 1)} C_1 \quad \text{and} \quad C_3 \geq \frac{|k_3|}{|k_1|} \frac{\Omega_1(N' + 1)}{\Omega_3(N' + 1)} C_1. \hspace{1cm} (10,11)
\]

The last inequality \(h_{1,j} \leq C_4 K^j\) is proved in the same way as \(h_{1,j} \leq C_1 K^j\) and, without going through all details, the induction step will work if

\[
\frac{1}{\Omega_4(N' + 1)} \sum_{m=1}^{N''-1} \left[ (2C_1 |k_4| + C_2 |k_5| + C_3 |k_6|) h_{1,m} \right.
\]

\[
+ (C_1 |k_5| + 2C_2 |k_7| + C_3 |k_8|) h_{2,m}
\]

\[
+ (C_1 |k_6| + C_2 |k_8| + 2C_3 |k_9|) h_{3,m} \right] K^{-m}
\]

\[
+ \left( C_1^2 |k_4| + C_1 C_2 |k_5| + C_1 C_3 |k_6| \right.
\]

\[
\left. + C_2^2 |k_7| + C_2 C_3 |k_8| + C_3^2 |k_9| \right) \frac{1}{|\lambda_4|} \leq C_4.
\]  \hspace{1cm} (12)
3.2. THE COORDINATE CHANGE FUNCTION $\phi$

These induction conditions are checked in the second part of the Maple program `phi_error.mpl` with $N'' = 19$ and the same parameter values as for the first part of the remainder term. It turns out that

- $h_{1,j} \leq 4.00 \times 10^{-4} \cdot 5^j$ for $j \geq 40$,
- $h_{2,j} \leq 4.59 \times 10^{-4} \cdot 5^j$ for $j \geq 40$,
- $h_{3,j} \leq 5.12 \times 10^{-3} \cdot 5^j$ for $j \geq 40$,
- $h_{4,j} \leq 5.80 \times 10^{-3} \cdot 5^j$ for $j \geq 40$.

A consequence is that the second part of the remainder term is considerably smaller than the first part. The details are listed in Table 3.2.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$R_1$</th>
<th>$R_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$3.23 \times 10^{-10}$</td>
<td>$3.64 \times 10^{-15}$</td>
</tr>
<tr>
<td>2</td>
<td>$1.06 \times 10^{-10}$</td>
<td>$4.17 \times 10^{-15}$</td>
</tr>
<tr>
<td>3</td>
<td>$6.75 \times 10^{-10}$</td>
<td>$4.66 \times 10^{-14}$</td>
</tr>
<tr>
<td>4</td>
<td>$1.75 \times 10^{-9}$</td>
<td>$5.28 \times 10^{-14}$</td>
</tr>
</tbody>
</table>

**Table 3.2:** This table lists upper bounds on the remainder terms $R_1$ and $R_2$ for the different components $i$. These bounds are calculated in the Maple program `phi_error.mpl`.

All this adds up to an estimate of the remainder term ($\phi^T$ denotes the truncated power series of $\phi$).

**Result 1.** If $\|z\|_{\infty} \leq \frac{1}{10}$ then $\|\phi(z) - \phi^T(z)\|_{\infty} \leq 1.75 \times 10^{-9}$.

If $\|z\|_{\infty} \leq \frac{1}{5}$ then $\|\phi(z) - \phi^T(z)\|_{\infty} \leq 1.50 \times 10^{-8}$.

**The derivatives of $\phi$**

We get the derivatives of $\phi$ by differentiating its power series

$$\frac{\partial \phi_i}{\partial z_k}(z) = \sum_j c_{i,j} \frac{\partial}{\partial z_k} z^j = \sum_j c_{i,j} j_k z^{j - e_k}$$

and calculate them by truncating their power series

$$\frac{\partial \phi_i}{\partial z_k}(z) = \sum_{|j| \leq N} c_{i,j} j_k z^{j - e_k} + \text{Remainder term}.$$  

The remainder terms can be estimated in the same way as before by subdividing them in two parts and using the estimates from previous subsections,

$$\|D\phi(z) - D\phi^T(z)\|_{\infty} = \max_i \sum_{k=1}^4 \left| \frac{\partial \phi_i}{\partial z_k}(z) - \frac{\partial \phi^T_i}{\partial z_k}(z) \right| \leq \max_i \sum_{k=1}^4 \sum_{|j| > N} |c_{i,j} j_k| z^{j - e_k}$$
\[ \leq \max_i \sum_{k=1}^{4} \sum_{j=N+1}^{\infty} \left( \frac{1}{10} \right)^{j-1} \sum_{|j|=j} j_k |c_{i,j}| \]

\[ = \max_i \sum_{j=N+1}^{\infty} j \left( \frac{1}{10} \right)^{j-1} \sum_{|j|=j} |c_{i,j}| \]

\[ \leq \max_i \left( \sum_{j=N+1}^{N'} j h_{i,j} \left( \frac{1}{10} \right)^{j-1} + C_i K \sum_{j=N'+1}^{\infty} j \left( \frac{K}{10} \right)^{j-1} \right). \]

When this bound is calculated in the Maple program \texttt{phi_error.mpl} we get the following result.

\textbf{Result 2.} If \( \|z\|_\infty \leq \frac{1}{10} \) then \( \|D\phi(z) - D\phi^T(z)\|_\infty \leq 3.44 \times 10^{-7} \).

For later use we also have the following estimates.

\textbf{Lemma 1.} If \( \|z\|_\infty \leq \frac{1}{8} \) then \( \|\phi(z) - z\|_\infty \leq 0.72 \|z\|_\infty^2 \). If \( \|z\|_\infty \leq \frac{1}{8} \) then \( \|\phi^T(z) - z\|_\infty \leq 0.72 \|z\|_\infty^2 \). If \( \|z\|_\infty \leq \frac{1}{8} \) then \( \|D\phi(z) - I\|_\infty \leq 1.44 \|z\|_\infty \). If \( \|z\|_\infty \leq \frac{1}{8} \) then \( \|D\phi^T(z) - I\|_\infty \leq 1.44 \|z\|_\infty \).

These inequalities are derived in the Maple program \texttt{phi_error.mpl} using the following formulas

\[ \|\phi(z) - z\|_\infty = \max_i |\phi_i(z) - z_i| \]

\[ = \max_i \left| \sum_{|j| \geq 2} c_{i,j} z^j \right| \]

\[ \leq \|z\|_\infty^2 \cdot \max_i \left[ \sum_{j=2}^{N'} h_{i,j} \left( \frac{1}{8} \right)^{j-2} + C_i \sum_{j=N'+1}^{\infty} j \left( \frac{K}{8} \right)^{j-2} \right] \]

and

\[ \|D\phi(z) - I\|_\infty = \max_i \sum_{k=1}^{4} \left| \frac{\partial \phi_i}{\partial z_k}(z) - \delta_{ik} \right| \]

\[ = \max_i \sum_{k=1}^{4} \sum_{|j| \geq 2} c_{i,j} j_k z^{-e_k} \]

\[ \leq \|z\|_\infty \cdot \max_i \left[ \sum_{j=2}^{N'} j h_{i,j} \left( \frac{1}{8} \right)^{j-2} + C_i \sum_{j=N'+1}^{\infty} j \left( \frac{K}{8} \right)^{j-2} \right]. \]
3.3. The coordinate change function $\psi$

In the last section we used the coordinate change $\phi$ to transform the Lorenz-Stenflo system to a normal form system and this resulted in a functional equation for $\phi$. By reversing the roles of the right-hand side functions $F$ and $G$ we get a corresponding functional equation for the inverse function $\psi$ of $\phi$,

$$D\psi(x)A x - A\psi(x) = G(\psi(x)) - D\psi(x)F(x).$$  \hspace{1cm} (*)

Our plan of how to calculate $\psi$ is the same as that for $\phi$. We write $\psi$ as a (formal) power series

$$\psi(x) = \sum_j e_{i,j} x^j$$

and use the functional equation to derive a recurrence equation for the coefficients $e_{i,j}$. The function $\psi$ can then be approximated by truncating its power series

$$\psi(x) = \sum_{|j| \leq N} e_{i,j} x^j + \text{Remainder term}.$$  

At this point we are faced with the problem of estimating the remainder term. If we compare the situation to the situation for $\phi$ there is an important difference and this can be traced back to the term $G(\psi(x))$ in the functional equation. In the case of the functional equation for $\phi$ we had the term $F(\phi(z))$ which we are able to handle since $F$ is only a second degree polynomial, but here $G$ is a full-blown power series and this means that there are difficulties in controlling the coefficients of $G(\psi(x))$. We will therefore not follow the same path we used for $\phi$ but instead use the fact that $\psi$ is the inverse function of $\phi$ and that we already have a good error bound on the truncated power series of $\phi$. If we let $\phi^T$ and $\psi^T$ denote the truncated power series of $\phi$ and $\psi$, respectively, then we will manage to show that the remainder term is small by basically using that $\|\phi^T - \phi\|_\infty$ and $\|\psi^T \circ \phi^T - \text{id}\|_\infty$ are small. Corresponding estimates of the derivatives of $\psi$ are also made.

**Recurrence equations**

When we insert the power series of $\psi$ into the functional equation (*) the left-hand side has the same form as the left-hand side of the functional equation for $\phi$,

$$\left(D\psi(x)A x - A\psi(x)\right)_{(i)} = \sum_j \left( \sum_{k=1}^{4} \lambda_k j_k - \lambda_i \right) e_{i,j} x^j.$$  

Turning to the right-hand side we must keep in mind that the power series of $G$ only contains terms of degree higher or equal to $2M$ and since $\psi$ has no constant term the composition $G(\psi(x))$ will also only have those higher degree terms. In
our case the degree $N$ of the truncation is lower than $2M$, so for the parameter range of the coefficients we are interested in the right-hand has only one term,

$$
(D\psi(x)F(x))_i = \sum_{k=1}^{4} \frac{\partial \psi_i}{\partial x_k}(x) F_k(x)
$$

$$
= \sum_{k=1}^{4} \left( \sum_{j} e_{i,j+k}(j_1+1)x^j \right) F_k(x)
$$

$$
= \left( \sum_{j} e_{i,j+1}(j_1+1)x^j \right) k_1(x_1 + x_2 + x_3)x_4
$$

$$+ \left( \sum_{j} e_{i,j+2}(j_2+1)x^j \right) k_2(x_1 + x_2 + x_3)x_4$$

$$+ \left( \sum_{j} e_{i,j+3}(j_3+1)x^j \right) k_3(x_1 + x_2 + x_3)x_4$$

$$+ \left( \sum_{j} e_{i,j+4}(j_4+1)x^j \right) \left( k_4x_1^2 + k_5x_1x_2 + k_6x_1x_3
$$

$$+ k_7x_2^2 + k_8x_2x_3 + k_9x_3^2 \right).$$

If we expand this expression, collect all terms of equal degree and identify with the left-hand side we get the following recurrence equation for the coefficients $e_{i,j}$,

$$
\Omega_i(j) e_{i,j} = -e_{i,j-x_4 j_1} k_1 - e_{i,j-e_1-e_2-e_4} (j_1+1)k_1
$$

$$- e_{i,j+e_1-e_3-e_4} (j_1+1)k_1 - e_{i,j-e_1+e_2-e_4} (j_2+1)k_2$$

$$- e_{i,j-e_1-e_2-e_4} (j_2+1)k_2$$

$$- e_{i,j-e_1+e_3-e_4} (j_3+1)k_3 - e_{i,j-e_2+e_3-e_4} (j_3+1)k_3$$

$$- e_{i,j-e_1-e_2+e_4} (j_3+1)k_3$$

$$- e_{i,j+e_1+e_2-e_4} (j_4+1)k_4$$

$$- e_{i,j-e_1+e_3+e_4} (j_4+1)k_5 - e_{i,j-e_1-e_3+e_4} (j_4+1)k_5$$

$$- e_{i,j+e_2+e_4} (j_4+1)k_7 - e_{i,j-e_2+e_3-e_4} (j_4+1)k_8$$

$$- e_{i,j-e_2-e_3+e_4} (j_4+1)k_9.$$
cube around the origin. This we are able to show by using Lemma 1 which we recapitulate here: \( \phi \) and \( \phi^T \) satisfy the following estimates

- \( \|\phi(z) - z\|_\infty \leq 0.72 \|z\|_\infty^2 \) for \( \|z\|_\infty \leq \frac{1}{8} \), \( (13) \)
- \( \|\phi^T(z) - z\|_\infty \leq 0.72 \|z\|_\infty^2 \) for \( \|z\|_\infty \leq \frac{1}{8} \), \( (14) \)
- \( \|D\phi(z) - I\|_\infty \leq 1.44 \|z\|_\infty \) for \( \|z\|_\infty \leq \frac{1}{8} \), \( (15) \)
- \( \|D\phi^T(z) - I\|_\infty \leq 1.44 \|z\|_\infty \) for \( \|z\|_\infty \leq \frac{1}{8} \), \( (16) \)

where \( \phi^T \) denotes the truncated power series of \( \phi \).

**Lemma 2.** The functions \( \phi \) and \( \phi^T \) are injective in the cube \( C_{1/8} = \{\|z\|_\infty \leq \frac{1}{8}\} \).

**Proof.** Pick two separate points \( z_1 \) and \( z_2 \) inside the cube. By using the triangle inequality, the mean value theorem and inequality (15) we have that

\[
\|\phi(z_1) - \phi(z_2)\|_\infty = \|[\phi(z_1) - z_1] - [\phi(z_2) - z_2] - (z_2 - z_1)\|_\infty \\
\geq \|z_1 - z_2\|_\infty - \|[\phi(z_1) - z_1] - [\phi(z_2) - z_2]\|_\infty \\
\geq \|z_1 - z_2\|_\infty - \|D\phi(\zeta) - I\|_\infty \cdot \|z_1 - z_2\|_\infty \\
= (1 - \|D\phi(\zeta) - I\|_\infty) \cdot \|z_1 - z_2\|_\infty > 0,
\]

where \( \zeta \) is a point on the line segment between \( z_1 \) and \( z_2 \) (hence inside the cube). This shows that \( \phi \) is injective. The proof for \( \phi^T \) is analogous.

Next we show that the images of \( \phi \) and \( \phi^T \) cover the cube \( C_{1/9} = \{\|z\| \leq \frac{1}{9}\} \) and together with the last lemma this shows that \( \psi = \phi^{-1} \) and \( (\phi^T)^{-1} \) are defined and analytical in that cube.

**Lemma 3.** If \( \|z\|_\infty = \frac{1}{8} \) then \( \|\phi(z)\|_\infty \geq \frac{1}{9} \) and \( \|\phi^T(z)\|_\infty \geq \frac{1}{9} \).

If \( \|z\|_\infty = \frac{1}{9} \) then \( \|\phi(z)\|_\infty \geq \frac{1}{10} \) and \( \|\phi^T(z)\|_\infty \geq \frac{1}{10} \).

**Proof.** When \( \|z\|_\infty = \frac{1}{8} \) we have that

\[
\|\phi(z)\|_\infty \geq \|z\|_\infty - \|\phi(z) - z\|_\infty \geq \frac{1}{8} - 0.72 \cdot (\frac{1}{8})^2 > \frac{1}{9}
\]

and when \( \|z\|_\infty = \frac{1}{9} \) we have that

\[
\|\phi(z)\|_\infty \geq \|z\|_\infty - \|\phi(z) - z\|_\infty \geq \frac{1}{9} - 0.72 \cdot (\frac{1}{9})^2 > \frac{1}{10}.
\]

The proof for \( \phi^T \) is analogous.

The estimate of the remainder term that we will use hinges on Result 1 in Section 3.2 and on the following two lemmas.

**Lemma 4.** If \( \|x\|_\infty \leq \frac{1}{6} \) then \( \|D\phi^{-1}(x)\|_\infty \leq 1.3 \).

If \( \|x\|_\infty \leq \frac{1}{6} \) then \( \|D(\phi^T)^{-1}(x)\|_\infty \leq 1.3 \).
Proof. Differentiating the identity $\phi \circ \phi^{-1}(x) = x$ using the chain rule gives us
\[ D\phi^{-1}(x) = [D\phi(\phi^{-1}(x))]^{-1}. \]

If we use that $\phi^{-1}(C_{1/9}) \subseteq C_{1/8}$ (Lemma 3) together with the following standard formula (see, for example, Lemma 2.3.3 in [Matrix Computations (1996), Johns Hopkins])
\[
\| (I - A)^{-1} \|_\infty \leq \frac{1}{1 - \| A \|_\infty} \quad \text{(valid when } \| A \|_\infty < 1) \]

we have that
\[
\| D\phi^{-1}(x) \|_\infty \leq \max_{z \in C_{1/8}} \| [D\phi(z)]^{-1} \|_\infty \leq \frac{1}{\max_{z \in C_{1/8}} \| D\phi(z) - I \|_\infty}
\leq \frac{1}{\max_{z \in C_{1/8}} 1.44 \| z \|_\infty} \leq 1.3,
\]

where we also have used inequality (15). The same proof works for $\phi^T$. \hfill \Box

Lemma 5. If $z \in C_{1/9}$ then $\| \psi^T \circ \phi^T(z) - z \|_\infty \leq 1.32 \times 10^{-10}$.
If $z \in C_{1/9}$ then $\| D(\psi^T \circ \phi^T)(z) - I \|_\infty \leq 1.07 \times 10^{-8}$.

Proof. We define coefficients $a_{i,j}$ by
\[
(\psi^T \circ \phi^T)_i(z) = \sum_j a_{i,j} z^j.
\]

The polynomials $\phi^T$ and $\psi^T$ are the truncations of degree $N$ of the power series of $\phi$ and $\psi$, respectively, which means that $\phi(z) = \phi^T(z) + O(r^{N+1})$ and $\psi(x) = \psi^T(x) + O(r^{N+1})$. If we use the relation $\psi \circ \phi(z) = z$ and that $\phi^T$ and $\psi^T$ do not have constant terms it is easy to see that $\psi^T \circ \phi^T(z) - z = O(r^{N+1})$, i.e. $a_{i,j} = 0$ if $|j| \leq N$ and $j \neq e_i$. Moreover, the composition $\psi^T \circ \phi^T(z)$ must be a polynomial of degree at most $N^2$.

The estimates that we want to prove are derived in the following way
\[
\| \psi^T \circ \phi^T(z) - z \|_\infty = \max_i \left| (\psi^T \circ \phi^T)_i(z) - z_i \right|
= \max_i \left| \sum_{N < |j| \leq N^2} a_{i,j} z^j \right|
\leq \sum_{j=N+1}^{N^2} \left( \max_i \sum_{|j|=j} |a_{i,j}| \left( \frac{1}{5} \right)^j \right),
\]
\[
\| D(\psi^T \circ \phi^T)(z) - I \|_\infty = \max_i \sum_{k=1}^4 \left| D_k(\psi^T \circ \phi^T)_i(z) - \delta_{ik} \right|
\]
3.3. **The Coordinate Change Function** \( \psi \)

\[
\psi_{109} = \max_i \left| \sum_{k=1}^{4} \sum_{N<|j| \leq N^2} a_{i,j} j_k z^{j-e_k} \right|
\]

\[
\leq \max_i \sum_{k=1}^{4} \sum_{N<|j| \leq N^2} |a_{i,j}| j_k |z^{j-e_k}|
\]

\[
\leq \max_i \sum_{N<|j| \leq N^2} |a_{i,j}| \left( \frac{1}{N} \right)^j \sum_{k=1}^{4} j_k
\]

\[
= \sum_{j=N+1}^{N^2} \left( \max_i \sum_{|j|=j} |a_{i,j}| \right) j \left( \frac{1}{N} \right)^{j-1}.
\]

From these formulas we see that the key to our estimates is to get a good bound on the quantities

\[
\max_i \sum_{|j|=j} |a_{i,j}|.
\]

The rest of the proof is therefore directed at estimating various coefficients that together lead to an estimate of the quantities above.

First we introduce the coefficients \( f_{n,j} \) as

\[
\phi_1^T(z)^{n_1} \phi_2^T(z)^{n_2} \phi_3^T(z)^{n_3} \phi_4^T(z)^{n_4} = \sum_j f_{n,j} z^j.
\]  

These coefficients have the following obvious boundary conditions: \( f_{n,j} = 0 \), if \( n \neq 0 \) or \( j \neq 0 \); \( f_{e_{i,j} = c_{i,j}} \) if \( |j| \leq N \); \( f_{n,j} = 1 \), if \( j = n \); \( f_{n,j} = 0 \), if \( |j| \leq |n| \) and \( j \neq n \). By multiplying (*) with \( \phi_1^T(z) \) and identifying coefficients we get a recurrence equation for the coefficients \( f_{j,n} \),

\[
f_{n+e_{i,j}} = \sum_{m:|j-m| \leq N} c_{i,j-m} f_{n,m}.
\]

With these coefficients we can find an expression for \( a_{i,j} \),

\[
\sum_j a_{i,j} z^j = (\psi^T \circ \phi^T)(z) = \sum_m e_{i,m} \phi_1^T(z)^{m_1} \phi_2^T(z)^{m_2} \phi_3^T(z)^{m_3} \phi_4^T(z)^{m_4}
\]

\[
= \sum_m e_{i,m} \sum_j f_{m,j} z^j
\]

\[
= \sum_j \left( \sum_m e_{i,m} f_{m,j} \right) z^j,
\]

which shows that

\[
a_{i,j} = \sum_m e_{i,m} f_{m,j}.
\]
We will now derive a sequence \( \{b_{n,j}\}_{n,j} \) of numbers that estimates the coefficients \( f_{n,j} \) in the following way

\[
b_{n,j} \geq \max_{|n|=n} \sum_{|j|=j} |f_{n,j}|.
\]

As boundary conditions we put \( b_{n,j} = 0 \) if \( j < n \) and \( b_{n,j} = 1 \) if \( j = n \), and by using the recurrence equation of \( f_{n,j} \) we have that

\[
\max_{|n|=n} \sum_{|j|=j} |f_{n,j}| \leq \max_{|n|=n} \sum_{|j|=j} |c_{i,j-m}| |f_{n-e_i,m}|
\]

\[
= \max_{|n|=n} \sum_{m=1}^{|j|=j} |f_{n-e_i,m}| \sum_{|j|=j} |c_{i,j-m}|
\]

\[
= \sum_{m=1}^{j-1} \max_{|n|=n} \sum_{|m|=m} |f_{n-e_i,m}| \sum_{|j|=j} |c_{i,j-m}|
\]

\[
\leq \sum_{m=1}^{j-1} b_{n-1,m} h_{i,j-m}.
\]

We therefore define

\[
b_{n,j} = \min_i \sum_{m=1}^{j-1} b_{n-1,m} h_{i,j-m} \text{ for } j \geq j_0
\]

and put

\[
b_{n,j} = \max_{|n|=n} \sum_{|j|=j} |f_{n,j}| \text{ for } j < j_0.
\]

Now we can estimate the sum over the coefficients \( a_{i,j} \),

\[
\max_i \sum_{|j|=j} |a_{i,j}| \leq \max_i \sum_{|m|=m} |e_{i,m}| \sum_{|j|=j} |f_{m,j}|
\]

\[
\leq \sum_{m=1}^{j} \max_{|m|=m} |f_{m,j}| \cdot \max_i \sum_{|m|=m} |e_{i,m}|
\]

\[
\leq \sum_{m=1}^{j} b_{m,j} p_m,
\]

where \( p_m = \max_i \sum_{|m|=m} |e_{i,m}|. \)

In the Maple program \texttt{psi_error.mpl} these estimates are performed and this yields the result stated in Lemma 5.

Now we are ready to estimate the remainder terms. We start with the remainder term in the truncation of \( \psi \).
3.3. THE COORDINATE CHANGE FUNCTION $\psi$

**Result 3.** If $\|x\|_\infty \leq \frac{1}{10}$ then $\|\psi(x) - \psi^T(x)\|_\infty \leq 1.97 \times 10^{-8}$.

**Proof.** Fixing $x$, we use the triangle inequality for vector norms

$$\|\psi(x) - \psi^T(x)\|_\infty \leq \|\psi(x) - (\phi^T)^{-1}(x)\|_\infty + \|(\phi^T)^{-1}(x) - \psi^T(x)\|_\infty.$$ 

The right-most term in the right-hand side can be estimated using Lemma 3 and 5,

$$\|(\phi^T)^{-1}(x) - \psi^T(x)\|_\infty \leq \max_{z \in (\phi^T)^{-1}(C_{1/10})} \|z - \psi^T \circ \phi^T(z)\|_\infty \leq \max_{z \in C_{1/9}} \|\psi^T \circ \phi^T(z) - z\|_\infty \leq 1.32 \times 10^{-10}.$$ 

After some rewriting, the left-most term in the right-hand side can be formulated as

$$\|\psi(x) - (\phi^T)^{-1}(x)\|_\infty = \|\phi^{-1}(x) - (\phi^T)^{-1}(x)\|_\infty \leq \max_{z \in \phi^{-1}(C_{1/10})} \|z - (\phi^T)^{-1} \circ \phi(z)\|_\infty \leq \max_{z \in \phi^{-1}(C_{1/10})} \|\phi^T(z) - \phi(z)\|_\infty \leq \|\phi^T(z) - \phi(z)\|_\infty.$$ 

where we have used the mean value theorem in the last step. Result 1 gives us that $\|w\|_\infty = \|\phi(z) - \phi^T(z)\|_\infty \leq 1.50 \times 10^{-8}$, which means that

$$\xi = \phi^T(z) + \tau (\phi(z) - \phi^T(z)) = \phi(z) + (1 - \tau)(\phi^T(z) - \phi(z))$$

is inside the cube $C_{1/10 + 1.50 \times 10^{-8}} \subset C_{1/9}$. We can now use Lemma 4 to complete the estimate:

$$\max_{z \in \phi^{-1}(C_{1/10})} \|D(\phi^T)^{-1}(\xi)w\|_\infty \leq \max_{\xi \in C_{1/9}} \|D(\phi^T)^{-1}(\xi)\|_\infty \cdot \|w\|_\infty \leq 1.95 \times 10^{-8}. \quad \Box$$

Before we address the remainder term of the derivative $D\psi$ we need an intermediate result that plays the same role in that proof as the usage of the mean value theorem does in the proof of Result 3.
Lemma 6. If \(\|x\|_\infty, \|y\|_\infty \leq \frac{1}{8}\) then \(\|D\phi^T(x) - D\phi^T(y)\|_\infty \leq 1.71 \|x - y\|_\infty\).

Proof. We start by looking at the expression
\[
\frac{\partial \phi_i^T}{\partial z_k}(x) - \frac{\partial \phi_i^T}{\partial z_k}(y) = \sum_{|j| \leq N} e_i.j \cdot j_k \cdot (x^j - y^j).
\]
and if we are to estimate this quantity we first need a bound on \(|x^j - y^j|\). By using the mean value theorem on the function \(f(x) = x^j\) we get that
\[
|x^j - y^j| \leq |j| \left( \max\{\|x\|_\infty, \|y\|_\infty\} \right)^{|j|-1} \|x - y\|_\infty
\]
Now we have that
\[
\|D\phi^T(x) - D\phi^T(y)\|_\infty \leq \max_i \sum_{k=1}^{4} \left| \frac{\partial \phi_i^T}{\partial z_k}(x) - \frac{\partial \phi_i^T}{\partial z_k}(y) \right|
\]
\[
\leq \max_i \sum_{j=1}^{N} (j-1)(\frac{1}{8})^{j-2} \|x - y\|_\infty \sum_{k=1}^{4} \sum_{|j| = j} j_k |e_i.j|
\]
\[
\leq \left[ \sum_{j=1}^{N} j(j-1)p_j(\frac{1}{8})^{j-2} \right] \|x - y\|_\infty,
\]
where \(p_j\) are the quantities we introduced in the proof of Lemma 5. In the Maple program \texttt{psi\_error.mpl}\ we calculate the factor in front of \(\|x - y\|_\infty\) to be less than 1.71. \(\square\)

Result 4. If \(\|x\|_\infty \leq \frac{1}{10}\) then \(\|D\psi(x) - D\psi^T(x)\|_\infty \leq 6.53 \times 10^{-7}\).

Proof. This proof uses the same strategy as the proof of Result 3. First we fix \(x\) and use the triangle inequality for matrix norms
\[
\|D\psi(x) - D\psi^T(x)\|_\infty \leq \|D\psi(x) - D(\phi^T)^{-1}(x)\|_\infty + \|D(\phi^T)^{-1}(x) - D\psi^T(x)\|_\infty.
\]
We estimate the right-most term in the right-hand side with Lemma 4 and 5,
\[
\|D(\phi^T)^{-1}(x) - D\psi^T(x)\|_\infty
\]
\[
= \|D(\phi^T)^{-1}(x) - D\psi^T(x)D\phi^T((\phi^T)^{-1}(x))D(\phi^T)^{-1}(x)\|_\infty
\]
\[
\leq \|I - D\psi^T(x)D\phi^T((\phi^T)^{-1}(x))\|_\infty \cdot \|D(\phi^T)^{-1}(x)\|_\infty
\]
\[
\leq 1.3 \cdot \|D(\psi^T \circ \phi^T)((\phi^T)^{-1}(x)) - I\|_\infty
\]
\[
\leq 1.3 \cdot \max_{z \in (\phi^T)^{-1}(C_{1/10})} \|D(\psi^T \circ \phi^T)(z) - I\|_\infty
\]
\[
\leq 1.3 \cdot \max_{z \in C_{1/10}} \|D(\psi^T \circ \phi^T)(z) - I\|_\infty \leq 1.40 \times 10^{-8}.
\]
3.3. THE COORDINATE CHANGE FUNCTION \( \psi \)

The left-most term in the right-hand side is dealt with the same type of approximations we used for the corresponding term in Result 3 except that we use Lemma 6 instead of the mean value theorem,

\[
\| D\psi(x) - D(\phi^T)^{-1}(x) \|_{\infty} \\
= \| D\phi^{-1}(x) - D(\phi^T)^{-1}(x) \|_{\infty} \\
= \| D\phi^{-1}(x) - D(\phi^T)^{-1}(x) \|_{\infty} D\phi^{-1}(x) \|_{\infty} \\
\leq \| I - D(\phi^T)^{-1}(x) D\phi(\phi^{-1}(x)) D\phi^{-1}(x) \|_{\infty} \cdot \| D\phi^{-1}(x) \|_{\infty} \\
\leq 1.3 \cdot \| I - D(\phi^T)^{-1}(x) [D\phi((\phi^T)^{-1}(x))] \|_{\infty} \\
+ (D\phi(\phi^{-1}(x)) - D(\phi^T)^{-1}(x))) \|_{\infty} \\
= 1.3 \cdot \| D(\phi^T)^{-1}(x) D\phi(\phi^{-1}(x)) - D(\phi^T)^{-1}(x)) \|_{\infty} \\
\leq 1.3 \cdot \| D(\phi^T)^{-1}(x) \|_{\infty} \cdot \| D\phi(\phi^{-1}(x)) - D(\phi^T((\phi^T)^{-1}(x))) \|_{\infty} \\
\leq 1.3^2 \cdot \| D\phi(\phi^{-1}(x)) - D(\phi^T)^{-1}(x)) \|_{\infty} \\
+ 1.3^2 \cdot \| D(\phi^T((\phi^T)^{-1}(x))) \|_{\infty} .
\]

The first term is estimated using Result 2 and we obtain that it is less or equal to

\[
1.3^2 \cdot \| D\phi(\phi^{-1}(x)) - D(\phi^T)^{-1}(x)) \|_{\infty} \\
\leq 1.3^2 \cdot \max_{z \in \phi^{-1}(C_{1/10})} \| D\phi(z) - D(\phi^T)^{-1}(x)) \|_{\infty} \\
\leq 1.3^2 \cdot \max_{z \in C_{1/9}} \| D\phi(z) - D(\phi^T(z)) \|_{\infty} \leq 5.82 \times 10^{-7},
\]

while for the second term we use that \( \| \phi^{-1}(x) - (\phi^T)^{-1}(x) \|_{\infty} \leq 1.95 \times 10^{-8} \) (from the proof of Result 3) and Lemma 6,

\[
1.3^2 \cdot \| D(\phi^T)^{-1}(x)) - D(\phi^T((\phi^T)^{-1}(x))) \|_{\infty} \\
\leq 1.3^2 \cdot 1.71 \cdot \| \phi^{-1}(x) - (\phi^T)^{-1}(x) \|_{\infty} \leq 5.64 \times 10^{-8}.
\]

We end this section with estimates of \( \psi^T \) and \( D\psi^T \), which are similar to the estimates appearing in Lemma 1.

**Lemma 7.** If \( \| x \|_{\infty} \leq \frac{1}{10} \) then \( \| \psi^T(x) - x \|_{\infty} \leq 0.8 \| x \|_{\infty}^2 \).

If \( \| x \|_{\infty} \leq \frac{1}{10} \) then \( \| D\psi^T(x) - I \|_{\infty} \leq 1.6 \| x \|_{\infty} \).

**Proof.** The proof uses these two estimates

\[
\| \psi^T(x) - x \|_{\infty} = \max_i \left| \sum_{2 \leq |j| \leq N} e_{i,j} x^j \right| \leq \| x \|_{\infty}^2 \cdot \max_i \left[ \sum_{j=2}^{N} p_j \left( \frac{1}{10} \right)^{j-2} \right]
\]

\[
\| D\psi^T(x) - I \|_{\infty} \leq 1.6 \| x \|_{\infty}.
\]
and

\[ \|D\psi^T(x) - I\|_\infty = \max_i \left| \sum_{k=1}^{4} \sum_{2 \leq |j| \leq N} e_{i,j,k} x^j e_k \right| \]
\[ \leq \|x\|_\infty \cdot \max_i \left[ \sum_{j=2}^{N} j p_j \left( \frac{1}{10} \right)^{j-2} \right]. \]

These two expressions are calculated in the program psi_estimates.mpl. \( \square \)

### 3.4 The non-linear function \( G \)

Let us review the role of \( G \) in the coordinate change. The function \( G \) is present in three places: in the coordinate functions \( \phi \) and \( \psi \), and in the normal form flow \( \dot{z} = Az + G(z) \). If we look at the coordinate changes \( \phi \) and \( \psi \) they are both calculated with truncated power series and these truncations are of low enough order not to be directly influenced by \( G \) (meaning that their terms are unaffected by changes to any of the non-zero coefficients of \( G \)). It is only when we estimate the remainder terms that \( G \) comes in. Also when we deal with the normal form flow the function \( G \) will only play a minor part since we will approximate this flow inside the cube with the linear flow \( \dot{z} = Az \) (see Section 3.5).

Our focus in this section will first be to show that the power series of \( G \) is convergent inside the cube. This will prove that the whole analytic change of coordinates exists. The second objective is to show that \( G \) is sufficiently small so that the approximation with the linear flow works.

The plan we follow is very much the same plan we used for estimating the remainder term of \( \phi \). We have already derived recurrence equations (5) to (8) for the coefficients of \( G \) and to estimate \( G \) we divide its power series into two parts

\[ G_i(z) = R_1 + R_2 = \sum_{2M \leq |j| \leq N'} d_{i,j} z^j + \sum_{|j| > N'} d_{i,j} z^j. \]

(Coefficients \( d_{i,j} \) of order \(|j| < 2M\) are zero.) By introducing a sequence \( \{g_{i,j}\} \) of numbers that bounds the sums

\[ g_{i,j} \geq \sum_{|j|=j} |d_{i,j}| \]

and are calculated with a simple recurrence equation we are able to bound the first part of \( G \) inside the cube \( \{|z_i| \leq \frac{1}{9}\} \) with

\[ |R_1| \leq |z_1|^M \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^M \sum_{2M \leq |j| \leq N'} g_{i,j} (\frac{1}{9})^{j-2M}, \]

where \( z = (z_1, z_2, z_3, z_4) \). We then use induction to prove an inequality of the form

\[ g_{i,j} \leq C_i' L^j, \quad \text{for } j > N', \]
3.4. **THE NON-LINEAR FUNCTION** $G$

where $L$ is slightly larger than the constant $K$ we used for $\phi$ (a trade-off to make the induction work). This results in an estimate of the second part of $G$

$$|R_2| \leq |z_1|^M (\max\{ |z_2|, |z_3|, |z_4| \})^M \sum_{j=N'+1}^{\infty} \frac{C'_i}{L^{2M}} (\frac{L}{9})^{j-2M}.$$ 

Besides giving us a bound of the form $\|G(z)\|_\infty \leq \kappa |z_1|^M (\max\{ |z_2|, |z_3|, |z_4| \})^M$ it also shows that the power series of $G$ is convergent inside the set $\|z\|_\infty < 1/L$.

**The first part of the estimate**

To find suitable numbers $g_{i,j}$ we use the same strategy as when we defined the sequence $\{h_{i,j}\}$. Take the absolute value on both sides of the recurrence equation (5) for $d_{1,j}$ and sum over the parameter set $J_i^c = \{ j \geq 0 : |j| = j \text{ and } j_1 \geq M \text{ and } j_2 + j_3 + j_4 \geq M \}$, we conclude that

$$LHS = \sum_{j \in J_i^c} |d_{1,j}| = \sum_{|j|=j} |d_{1,j}| = g_{1,j},$$

$$RHS \leq |k_1| \sum_m |c_{1,m} + c_{2,m} + c_{3,m}| \sum_{|j|=j} |c_{4,j-m}|$$

$$+ \sum_{|m| \geq 1} \sum_{k=1}^{4} |c_{1,m+e_k}|(m_k + 1) \sum_{|j|=j} |d_{k,j-m}|$$

$$\leq |k_1| \sum_{m=1}^{j-1} h_{4,j-m} \sum_{|m|=m} (|c_{1,m}| + |c_{2,m}| + |c_{3,m}|)$$

$$+ \sum_{m=1}^{j-2M} \sum_{k=1}^{4} g_{k,j-m} \sum_{|m|=m} |c_{1,m+e_k}|(m_k + 1)$$

$$\leq |k_1| \sum_{m=1}^{j-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m})$$

$$+ \sum_{m=1}^{j-2M} h_{1,m+1}(m + 1)(g_{1,j-m} + g_{2,j-m} + g_{3,j-m} + g_{4,j-m}).$$

We can therefore define

$$g_{1,j} = |k_1| \sum_{m=1}^{j-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m})$$

$$+ \sum_{m=1}^{j-2M} (m + 1)h_{1,m+1}(g_{1,j-m} + g_{2,j-m} + g_{3,j-m} + g_{4,j-m})$$ (17)
for $j > P$. Similar calculations make it natural to define

$$g_{2,j} = \sum_{m=1}^{j-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m})$$
$$+ \sum_{m=1}^{j-2M} (m+1)h_{2,m+1}(g_{1,j-m} + g_{2,j-m} + g_{3,j-m} + g_{4,j-m}), \tag{18}$$

$$g_{3,j} = \sum_{m=1}^{j-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m})$$
$$+ \sum_{m=1}^{j-2M} (m+1)h_{3,m+1}(g_{1,j-m} + g_{2,j-m} + g_{3,j-m} + g_{4,j-m}), \tag{19}$$

$$g_{4,j} = \sum_{m=1}^{j-1} (|k_4|h_{1,m}h_{1,j-m} + |k_5|h_{1,m}h_{2,j-m} + |k_6|h_{1,m}h_{3,j-m}$$
$$+ |k_7|h_{2,m}h_{2,j-m} + |k_8|h_{2,m}h_{3,j-m} + |k_9|h_{3,m}h_{3,j-m})$$
$$+ \sum_{m=1}^{j-2M} (m+1)h_{4,m+1}(g_{1,j-m} + g_{2,j-m} + g_{3,j-m} + g_{4,j-m}). \tag{20}$$

The second part of the estimate

If we start to analyse the recurrence equations (17) to (20) we notice that the right-hand sides consist of two sums. The first sum is very similar to the right-hand side of the corresponding recurrence equation for $h_{i,j}$ and the second sum is

$$\sum_{m=1}^{j-2M} (m+1)h_{i,m+1}(g_{1,j-m} + g_{2,j-m} + g_{3,j-m} + g_{4,j-m}).$$

This together with the fact that we were able to show that $h_{i,j}$ is bounded by $C_iK^j$ suggests that we can expect a similar bound on $g_{i,j}$, but the absence of the dampening factor $\Omega_i(j)$ in the first sum and the presence of the factor $m+1$ in the second sum means that an Ansatz of the form $g_{i,j} \leq C'_i K^j$ will fail. We will instead choose a slightly larger $L = 7$ and choose a bound in the form $g_{i,j} \leq C'_i L^j$. This gives us the edge we need.

To prove by induction that $g_{i,j} \leq C'_i L^j$ for $j > N'$ we assume that

$$g_{i,m} \leq C'_i L^m \quad \text{for } N'' \leq m \leq j - 1, \tag{*}$$

where we also assume that $2N'' \leq N' - 1$ (for the same reason as before when we
3.4. THE NON-LINEAR FUNCTION $G$

worked with $h_{i,j}$ and we want to show that

$$g_{1,j} \equiv |k_1| \sum_{m=1}^{j-1} h_{4,j-m}(h_{1,m} + h_{2,m} + h_{3,m}) + \sum_{m=1}^{j-2M} (m+1)h_{1,m+1}(g_{1,j-m} + g_{2,j-m} + g_{3,j-m} + g_{4,j-m}) \leq C_1'L^j.$$  

Following in the footprints of the proof of $h_{i,j} \leq C_i K^j$ we split the sums in the expression above in three parts

$$\sum_{m=1}^{j-1} = \sum_{m=1}^{N''-1} + \sum_{m=N''}^{j-N''} + \sum_{m=j-N''+1}^{j-1}$$

and use the induction hypothesis (*),

$$g_{1,j} \leq |k_1| \sum_{m=1}^{N''-1} C_4 K^{j-m} (h_{1,m} + h_{2,m} + h_{3,m}) + |k_1| \sum_{m=N''}^{j-N''} C_4 K^{j-m} (C_1 + C_2 + C_3) K^m + |k_1| \sum_{m=j-N''+1}^{j-1} h_{4,j-m} (C_1 + C_2 + C_3) K^m$$

$$+ \sum_{m=1}^{N''-1} (m+1)h_{1,m+1}(C_1' + C_2' + C_3' + C_4') L^{j-m} + \sum_{m=N''}^{j-N''} (m+1)C_1 K^{m+1} (C_1' + C_2' + C_3' + C_4') L^{j-m} + \sum_{m=j-N''+1}^{j-2M} (m+1)C_1 K^{m+1} \times (g_{1,j-m} + g_{2,j-m} + g_{3,j-m} + g_{4,j-m}).$$

This sizable formula can be rearranged and by factoring out $L^j$ be rewritten as

$$\frac{g_{1,j}}{L^j} \leq |k_1| C_4 \left(\frac{K}{L}\right)^j \sum_{m=1}^{N''-1} (h_{1,m} + h_{2,m} + h_{3,m}) K^{-m} + |k_1| C_4 (C_1 + C_2 + C_3) (j-2N''+1) \left(\frac{K}{L}\right)^j$$

$$+ |k_1| (C_1 + C_2 + C_3) \left(\frac{K}{L}\right)^j \sum_{m=1}^{N''-1} h_{4,m} K^{-m}.$$
\[
+ (C'_1 + C'_2 + C'_3 + C'_4) \sum_{m=1}^{N''-1} (m+1)h_{1,m+1}L^{-m} \\
+ C_1(C'_1 + C'_2 + C'_3 + C'_4)L \left( \frac{K}{L} \right)^{N''} \\
\quad \times \sum_{m=1}^{j-2N''+1} (m+N')(\frac{K}{L})^m \\
+ C_1 \left( \frac{K}{L} \right)^j \sum_{m=2M}^{N''-1} (j-m+1) \\
\quad \times (g_{1,m} + g_{2,m} + g_{3,m} + g_{4,m})K^{-(m-1)}.
\]

If the right-hand side is smaller than \( C'_1 \) then the induction will work. Instead of testing this condition for all values of \( j \) we can derive a single condition by substituting \( j \) for \( N' \) or \( \infty \) whenever this gives a dominating (finite) expression and in two places make non-obvious substitutions to get something bigger:

- in \((j - 2N'' + 1)\left( \frac{K}{L} \right)^j\) we put \( j = 2N'' - 1 - 1/\log(K/L) \), and
- in \((j - m + 1)\left( \frac{K}{L} \right)^j\) we put \( j = N' \).

Doing the same analysis for \( g_{2,j}, g_{3,j} \) and \( g_{4,j} \) we derive analogous induction conditions. In the Maple program \( g\_estimates.mpl \) we check these conditions and manage to prove that

- \( g_{1,j} \leq 2.77 \times 10^{-7} \cdot 7^j \) for \( j > N \),
- \( g_{2,j} \leq 4.73 \times 10^{-8} \cdot 7^j \) for \( j > N \),
- \( g_{3,j} \leq 3.39 \times 10^{-7} \cdot 7^j \) for \( j > N \),
- \( g_{4,j} \leq 7.79 \times 10^{-7} \cdot 7^j \) for \( j > N \).

This results in the following estimate.

**Result 5.** If \( \|z\|_{\infty} \leq \frac{1}{9} \) then \( \|G(z)\|_{\infty} \leq \kappa |z_1|^M (\max\{|z_2|, |z_3|, |z_4|\})^M \), where \( \kappa = 2.81 \times 10^3 \).

As we stated in the beginning of this section the constant \( \kappa \) is calculated with the formula

\[
\kappa = \max_i \left[ \sum_{j=2M}^{N'} g_{i,j} \left( \frac{1}{9} \right)^{j-2M} + \sum_{j=N'+1}^{\infty} \frac{C'_i L^{2M}}{9} \left( \frac{L}{9} \right)^{j-2M} \right]
\]

and this is done in the Maple program \( g\_estimates.mpl \). The estimate in Result 5 is derived by factoring out powers of \( |z_1| \) and \( \max\{|z_2|, |z_3|, |z_4|\} \) from the power
3.4. THE NON-LINEAR FUNCTION $G$

series of $G$. These common factors arise due to the fact that $d_{i,j} = 0$ when $j_1 < M$ or $j_2 + j_3 + j_4 < M$,

$$|G_i(z)| \leq \sum_j |d_{i,j}| \cdot |z^j|$$

$$\leq |z_1|^M \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^M \sum_j |d_{i,j}| |z_1|^j \cdot |z_1|^j \cdot |z_1|^j \cdot |z_1|^j$$

Then we replace $|z_1|, |z_2|, |z_3|$ and $|z_4|$ with $\frac{1}{9}$ inside the summation

$$|G_i(z)| \leq |z_1|^M \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^M \sum_j |d_{i,j}| \left( \frac{1}{9} \right)^{-M}$$

We end this section by deriving a corresponding estimate of the derivative $DG$.

**Result 6.** If $\|z\|_\infty \leq \frac{1}{9}$ then $\|DG(z)\|_\infty \leq \kappa' |z_1|^{M-1} \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^{M-1}$ where $\kappa' = 6.87 \times 10^3$.

**Proof.** First we estimate the individual components of $DG$ by factoring out common powers of the variables $z_1, z_2, z_3$ and $z_4$ from the power series,

$$\left| \frac{\partial G_i}{\partial z_1} (z) \right| \leq |z_1|^{M-1} \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^{M-1}$$

$$\times \sum_j |d_{i,j}| \cdot |z_1|^j \cdot |z_1|^j \cdot |z_1|^j \cdot |z_1|^j$$

and for $k = 2, 3$ or $4$,

$$\left| \frac{\partial G_i}{\partial z_k} (z) \right| \leq |z_1|^{M-1} \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^{M-1}$$

$$\times \sum_j |d_{i,j}| \cdot |z_1|^j \cdot |z_1|^j \cdot |z_1|^j \cdot |z_1|^j \cdot |z_1|^j \cdot |z_1|^j \cdot |z_1|^j$$

Since $\|z\|_\infty \leq \frac{1}{9}$ we can replace $|z_1|, |z_2|, |z_3|$ and $|z_4|$ with $\frac{1}{9}$ inside the summation
and get the following estimate

\[ \|G(z)\|_\infty \leq \max_i \sum_{k=1}^4 \left| \frac{\partial G_i}{\partial z_k}(z) \right| \]

\[ \leq |z_1|^{M-1} \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^{M-1} \max_i \sum_j \left| j \left( \frac{1}{9} \right)^{j-2M+1} d_{i,j} \right| \]

\[ \leq |z_1|^{M-1} \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^{M-1} \max_{j=2M} \sum_{j=2M}^{\infty} j \left( \frac{1}{9} \right)^{j-2M+1} g_{i,j}. \]

This shows that the constant \( \kappa' \) can be chosen to be

\[ \kappa' = \max_i \left[ \sum_{j=2M}^{N'} j \left( \frac{1}{9} \right)^{j-2M+1} g_{i,j} + \frac{C_i'}{L^{2M-1}} \sum_{j=N'+1}^{\infty} j \left( \frac{L}{9} \right)^{j-2M+1} \right], \]

which is calculated in the Maple program \texttt{g_estimates.mpl}. \hfill \Box

### 3.5 The normal form flow

The last and remaining part of our study in this chapter focus on analysing orbits of the normal form flow \( \dot{z} = Az + G(z) \) inside the cube \( C_{1/9} \). We have arranged things so that the non-linear term \( G(z) \) satisfies

\[ \|G(z)\|_\infty \leq \kappa |z_1|^M \left( \max\{|z_2|, |z_3|, |z_4|\} \right)^M \]

throughout the cube and this means that \( G \) is not only small but, more importantly, decreasing rapidly close to the stable and unstable manifolds. As we shall see this means that orbits of the system can be approximated well by the corresponding orbits of the linear system \( \dot{z} = Az \). The reason this works is that if an orbit stays a long time inside the cube (and, hence, the overall smallness of \( G \) is not a good enough approximation) it must spend most of that time close to the stable and unstable manifolds, in a region where \( G \) is falling-off rapidly, and this limits the accumulated error in the approximation. The initial set-up is that after a stepwise numerical integration (see Section 2.5 in Chapter 2) we have points inside the strip \( \{ x \in C_{1/10} : |x_2|, |x_3| \leq 10^{-5} \text{ and } x_4 = \frac{1}{10} \} \) on top of the cube \( C_{1/10} \). These points are first transformed with the change of coordinates \( z = \psi^T(x) \), and using the two inequalities (see Lemma 1)

\[ |\psi_i^T(x)| \leq |x_i| + |\psi_i^T(x) - x_i| \leq |x_i| + 0.8 \|x\|_\infty^2 \quad \text{and} \]

\[ |\psi_i^T(x)| \geq |x_i| - |\psi_i^T(x) - x_i| \geq |x_i| - 0.8 \|x\|_\infty^2 \]

we can see that the points are then inside the \( z \)-cube \( C_{1/9} \) and with \( |z_2|, |z_3| < z_4 \).

We will in this section show that for an orbit that starts in a point \( z \) in \( C_{1/9} \) with \( |z_2|, |z_3| < z_4 \) the difference between the orbit \( \varphi(z, t) \) of the normal form system
and the orbit $e^{A_t} \varphi(z,0)$ of the linear system is sufficiently small when the orbits exit the cube at $z_1 = \pm \frac{1}{10}$. To begin with, we derive estimates for the quantities $\varphi_1(z,t)$, $\varphi_4(z,t)$ and the exit time $\tau = \tau(z)$ (i.e. the time $\tau$ when $\varphi_1(z,\tau) = \pm \frac{1}{10}$). This is done by successively refining our estimates starting from a set of initial estimates. With these things in place it is then a matter of using the estimate of $G$ from the last section to get good bounds on $\|\varphi(z,\tau) - e^{A_\tau} \varphi(z,0)\|_\infty$ and $\|D\varphi(z,\tau) - e^{A_\tau}\|_\infty$, where $\tilde{\tau}$ is the exit time of the linear flow (i.e. the time $\tilde{\tau}$ when $e^{\lambda_1 \tilde{\tau}} \varphi_1(z,0) = \pm \frac{1}{10}$).

Henceforth we will assume, for simplicity, that $\varphi_1(z,0) > 0$ and therefore that orbits exit the cube at $x_1 = \frac{1}{10}$. The case $\varphi_1(z,0) < 0$ is symmetrical.

**Lemma 8.** If $\varphi_1(z,0) > 0$ then $\varphi_1(z,0) e^{(\lambda_1 - \kappa'')t} \leq \varphi_1(z,t) \leq \varphi_1(z,0) e^{(\lambda_1 + \kappa'')t}$, where $\kappa'' = 8.96 \times 10^{-8}$.

**Proof.** We have that $\dot{\varphi}_1(z,t) = \lambda_1 \varphi_1(z,t) + G_1(\varphi(z,t))$ and since the non-linear term satisfies $|G_1(z)| \leq \kappa \left(\frac{r}{B}\right)^{2M-1} z_1 \leq 8.96 \times 10^{-8} z_1$ we get that

$$(\lambda_1 - \kappa'')\varphi_1(z,t) \leq \varphi_1(z,t) \leq (\lambda_1 + \kappa'')\varphi_1(z,t).$$

The lemma then follows by applying Grönwall’s lemma.

Lemma 8 immediately gives us an estimate of the exit time $\tau$,

$$\frac{1}{\lambda_1 + \kappa''} \log \frac{r}{\varphi_1(z,0)} \leq \tau \leq \frac{1}{\lambda_1 - \kappa''} \log \frac{r}{\varphi_1(z,0)}$$

where $r = \frac{1}{10}$. From this it is also apparent that the exit time tends to infinity when the first coordinate of the starting point approaches the stable manifold.

**Lemma 9.** If $|\varphi_2(z,0)|, |\varphi_3(z,0)| < \varphi_4(z,0)$ then $|\varphi_2(z,t)|, |\varphi_3(z,t)| < \varphi_4(z,t)$ as long as the orbit stays inside the cube $C_{1/9}$.
Proof. If for some \( t > 0 \) we have that \( \varphi_i(z, t) = \varphi_4(z, t) \), for \( i = 2, 3 \), then we get
\[
\dot{\varphi}_i(z, t) = \lambda_i \varphi_i(z, t) + G_i(\varphi(z, t))
\leq \lambda_i \varphi_i(z, t) + \kappa |\varphi_1(z, t)|^M |\varphi_4(z, t)|^M
\leq \lambda_i \varphi_4(z, t) + \kappa \left( \frac{1}{9} \right)^M \left( \frac{1}{5} \right)^{M-1} \varphi_4(z, t),
\]
and
\[
\dot{\varphi}_4(z, t) = \lambda_4 \varphi_4(z, t) + G_4(\varphi(z, t))
\geq \lambda_4 \varphi_4(z, t) - \kappa |\varphi_1(z, t)|^M |\varphi_4(z, t)|^M
\geq \lambda_4 \varphi_4(z, t) - \kappa \left( \frac{1}{9} \right)^M \left( \frac{1}{5} \right)^{M-1} \varphi_4(z, t),
\]
which shows that
\[
\dot{\varphi}_4(z, t) - \dot{\varphi}_i(z, t) \geq [\lambda_4 - \lambda_i - 2\kappa \left( \frac{1}{9} \right)^{2M-1}] \varphi_4(z, t) > 0.
\]
If \( \varphi_i(z, t) = -\varphi_4(z, t) \) the same type of calculations show that \( \dot{\varphi}_4(z, t) + \dot{\varphi}_i(z, t) > 0 \). This means that the set \( \{ z \in C_{1/9} : -z_4 \leq z_2, z_3 \leq z_4 \} \) is a trapping region and the lemma follows. \( \square \)

For the orbits we focus on, Lemma 9 implies that
\[
\max\{|\varphi_2(z, t)|, |\varphi_3(z, t)|, |\varphi_4(z, t)|\} = \varphi_4(z, t)
\]
and the estimate we have on \( G \) can therefore be simplified to
\[
\|G(\varphi(z, t))\|_{\infty} \leq \kappa \varphi_1(z, t)^M \varphi_4(z, t)^M.
\]
We can now use the same type of reasoning as in Lemma 8 to get a bound on \( \varphi_4(z, t) \).

**Lemma 10.** If \( \varphi_1(z, 0) > 0 \) then \( \varphi_4(z, 0)e^{(\lambda_4 - \kappa')t} \leq \varphi_4(z, t) \leq \varphi_4(z, 0)e^{(\lambda_4 + \kappa')t} \).

Lemma 8 and 10 will serve as initial estimates for \( \varphi_1(z, t) \) and \( \varphi_4(z, t) \), and to improve these estimates we need the following generalisation of Grönwall’s lemma (inequalities of this type are discussed in Chapter 12 of \[Inequalities Involving Functions and Their Integrals and Derivatives\] (1991, Kluwer)).

**Lemma 11.** Let \( u(t) \) be a positive function that satisfies
\[
\dot{u}(t) \leq au(t) + f(t)u(t)^M, \quad \text{for } t \geq 0,
\]
where \( f(t) \) is a non-negative function and the following inequality holds
\[
(M - 1)u(0)^{M-1} \int_0^T f(s)e^{a(M-1)s} \, ds < 1,
\]
then we have for \( 0 \leq t \leq T \) that
\[
u(t) \leq u(0)e^{at} \left[ 1 - (M - 1)u(0)^{M-1} \int_0^t f(s)e^{a(M-1)s} \, ds \right]^{-\frac{1}{M-1}}.
\]
3.5. THE NORMAL FORM FLOW

Proof. Move the term \( au(t) \) over to the left-hand side of the differential inequality, divide both sides by \( u(t)^M \) and multiply by the integrating factor \(- (M-1)e^{a(M-1)t} \). The left-hand side can then be written as a derivative

\[
\frac{d}{dt} \left[ \frac{e^{a(M-1)t}}{u(t)^{M-1}} \right] \geq -(M-1)f(t)e^{a(M-1)t}.
\]

After integration we get the inequality

\[
\frac{e^{a(M-1)t}}{u(t)^{M-1}} - \frac{1}{u(0)^{M-1}} \geq -(M-1) \int_0^t f(s)e^{a(M-1)s} ds
\]

which can be rearranged to the inequality in the lemma. \( \square \)

**Corollary 12.** Let \( u(t) \) be a positive function that satisfies

\[
\dot{u}(t) \geq au(t) - f(t)u(t)^M, \quad \text{for } t \geq 0,
\]

where \( f(t) \) is a non-negative function, then we have for \( t \geq 0 \) that

\[
u(t) \geq u(0)e^{at} \left[ 1 + (M-1)u(0)^{M-1} \int_0^t f(s)e^{a(M-1)s} ds \right] - \frac{1}{M-1}.
\]

We will now use Lemma 11 and Corollary 12 to derive more precise estimates of the two flow variables \( \varphi_1(z,t) \) and \( \varphi_4(z,t) \).

**Lemma 13.** If \( \varphi_1(z,0) > 0 \) then

\[
(1 - \kappa_1 e^{M(\lambda_4 + \kappa'') t}) \varphi_1(z,0) e^{\lambda_1 t} \leq \varphi_1(z,t) \leq (1 + \kappa_1 e^{M(\lambda_4 + \kappa'') t}) \varphi_1(z,0) e^{\lambda_1 t}
\]

for \( 0 \leq t \leq T = 1/(\lambda_1 - \kappa'') \cdot \log(r/\varphi_1(z,0)) \), where \( \kappa_1 = 1.31 \times 10^{-9} \) and \( \kappa'' = 1.80 \times 10^{-7} \).

Proof. By using Lemma 10 we have that

\[
|G_1(\varphi(z,t))| \leq \kappa \varphi_4(z,0)^M e^{M(\lambda_4 + \kappa'') t} \varphi_1(z,t)^M,
\]

which shows that

\[
\dot{\varphi}_1(z,t) \leq \lambda_1 \varphi_1(z,t) + \kappa \varphi_4(z,0)^M e^{M(\lambda_4 + \kappa'') t} \varphi_1(z,t)^M.
\]
Before we can apply Lemma 11 to this differential inequality we first need to show that the inequality condition in Lemma 11 holds,

\[(M - 1) \varphi_1(z, 0)^{M-1} \int_0^T \kappa \varphi_4(z, 0)^M e^{M(\lambda_4 + \kappa'')s} e^{\lambda_1(M-1)s} ds \]

\[= \kappa(M - 1) \varphi_1(z, 0)^{M-1} \varphi_4(z, 0)^M \int_0^T e^{(M-1)\lambda_1 + M(\lambda_4 + \kappa'')s} ds \]

\[\leq \frac{\kappa(M - 1) \varphi_4(z, 0)^M}{(M - 1)\lambda_1 + M(\lambda_4 + \kappa'')} \left[ \varphi_1(z, 0)e^{\lambda_1\tilde{\tau}} \right]^{M-1} \times \exp\left[ (M(\lambda_4 + \kappa'')(1 + \varepsilon) + (M - 1)\lambda_1\varepsilon)\tilde{\tau} \right]. \]

We can write \( T = (1 + \varepsilon)\tilde{\tau} \), where \( \varepsilon = \kappa''/(\lambda_1 - \kappa'') \leq 7.92 \times 10^{-9} \), and since \( \varphi_1(z, 0)e^{\lambda_1\tilde{\tau}} = r \) and \( \varphi_4(z, 0) \leq \frac{1}{5} \) the right-hand side can be bounded by

\[\frac{\kappa(M - 1) \left( \frac{1}{5} \right)^M r^{M-1}}{(M - 1)\lambda_1 + M(\lambda_4 + \kappa'')} \exp\left[ (M(\lambda_4 + \kappa'')(1 + \varepsilon) + (M - 1)\lambda_1\varepsilon)\tilde{\tau} \right]. \]

The exponent satisfies \( M(\lambda_4 + \kappa'')(1 + \varepsilon) + (M - 1)\lambda_1\varepsilon \leq -15.9 < 0 \) which means that the exponential factor is smaller than 1 and the right-hand side is smaller than

\[\frac{\kappa(M - 1) \left( \frac{1}{5} \right)^M r^{M-1}}{(M - 1)\lambda_1 + M(\lambda_4 + \kappa'')} \leq 6.52 \times 10^{-8} < 1. \]

Lemma 11 now gives us that \( \varphi_1(z, t) \) is majorized by

\[\varphi_1(z, 0)e^{\lambda_1t} \frac{1}{\left[ 1 - \kappa(M - 1) \varphi_1(z, 0)^{M-1} \varphi_4(z, 0)^M \int_0^t e^{(M-1)\lambda_1 + M(\lambda_4 + \kappa'')s} ds \right]^{M-1}} \]

and after the integral in the denominator is evaluated this expression can be estimated by

\[\varphi_1(z, 0)e^{\lambda_1t} \frac{1}{\left[ 1 - \frac{\kappa(M - 1) \varphi_1(z, 0)^{M-1} \varphi_4(z, 0)^M e^{M(\lambda_4 + \kappa'')t}}{(M - 1)\lambda_1 + M(\lambda_4 + \kappa'')} \right]^{M-1}}. \]

The fact that \( t \leq (1 + \varepsilon)\tilde{\tau} \) and \( \varphi_1(z, 0)e^{\lambda_1\tilde{\tau}} = r \) means that

\[\left[ \varphi_1(z, 0)e^{\lambda_1t} \right]^{M-1} e^{M(\lambda_4 + \kappa'')t} \leq r^{M-1} e^{M(\lambda_4 + \kappa'')t}, \]

where \( \kappa''' = \kappa'' + \lambda_1\varepsilon \leq 1.80 \times 10^{-7} \), and we arrive at

\[\varphi_1(z, t) \leq \frac{\varphi_1(z, 0)e^{\lambda_1t}}{\left[ 1 - \frac{\kappa(M - 1) \left( \frac{1}{5} \right)^M r^{M-1} e^{M(\lambda_4 + \kappa'')t}}{(M - 1)\lambda_1 + M(\lambda_4 + \kappa'')} \right]^{M-1}}. \]
This expression can be further reduced by using the following elementary inequality

\[ \frac{1}{(1-y)^\alpha} \leq 1 + \frac{\alpha}{1-c} y \quad \text{for } 0 \leq y \leq c < 1 \text{ and } 0 \leq \alpha < 1, \]

which implies that \( \varphi_1(z, t) \) is bounded by

\[ \varphi_1(z, 0)e^{\lambda_1 t} \left[ 1 + \frac{\kappa \left( \frac{1}{5} \right)^M r^{M-1}}{(M-1)\lambda_1 + M(\lambda_1 + \kappa'') - \kappa(M-1)\left( \frac{1}{5} \right)^M r^{M-1}} e^{M(\lambda_4 + \kappa'') t} \right]. \]

To show the lower bound in the lemma we use the inequality

\[ \varphi_1(z, t) \geq \lambda_1 \varphi_1(z, t) - \kappa \varphi_4(z, 0)^M e^{M(\lambda_4 + \kappa'') t} \varphi_1(z, t)^M \]

and apply Corollary 12 in the same way.

We can use Lemma 13 to get a new estimate of the exit time. The inequality

\[ (1 - \kappa_1) \varphi_1(z, 0)e^{\lambda_1 t} \leq \varphi_1(z, t) \leq (1 + \kappa_1) \varphi_1(z, 0)e^{\lambda_1 t} \]

gives us that

\[ \tilde{\tau} - \frac{1}{\lambda_1} \log(1 + \kappa_1) \leq \tau \leq \tilde{\tau} - \frac{1}{\lambda_1} \log(1 - \kappa_1). \]

With this exit time estimate and the estimate in Lemma 13 we can use the same type of calculations as in the proof of Lemma 13 to get a new estimate of \( \varphi_4(z, t) \).

**Lemma 14.** If \( \varphi_1(z, 0) > 0 \) then

\[ (1 - \kappa_2 e^{(M-1)\lambda_4 t}) \varphi_4(z, 0)e^{\lambda_4 t} \leq \varphi_4(z, t) \leq (1 + \kappa_2 e^{(M-1)\lambda_4 t}) \varphi_4(z, 0)e^{\lambda_4 t} \]

for \( 0 \leq t \leq T = \tilde{\tau} - (1/\lambda_1) \log(1 - \kappa_1) \) and where \( \kappa_2 = 8.72 \times 10^{-9} \).

Although Lemma 13 is a good enough estimate for our needs we repeat the estimate calculations in the proof of that lemma once more and this time we use the estimate of \( \varphi_4 \) from Lemma 14 to eliminate the annoying coefficient \( \kappa'' \) and get a better looking estimate of \( \varphi_1(z, t) \).

**Lemma 15.** If \( \varphi_1(z, 0) > 0 \) then

\[ (1 - \kappa_3 e^{M\lambda_4 t}) \varphi_1(z, 0)e^{\lambda_1 t} \leq \varphi_1(z, t) \leq (1 + \kappa_3 e^{M\lambda_4 t}) \varphi_1(z, 0)e^{\lambda_1 t} \]

for \( 0 \leq t \leq T = \tilde{\tau} - (1/\lambda_1) \log(1 - \kappa_1) \) and where \( \kappa_3 = 1.31 \times 10^{-9} \).

We are now prepared to start working on the main estimate that compares \( \varphi(z, \tau) \) with \( e^{A\tau} \varphi(z, 0) \). By using the triangle inequality we have that

\[ \| \varphi(z, \tau) - e^{A\tau} \varphi(z, 0) \|_\infty = \max_{i=2,3,4} \| \varphi_i(z, \tau) - e^{\lambda_i \tau} \varphi_i(z, 0) \| \]

\[ \leq \max_{i=2,3,4} \| \varphi_i(z, \tau) - e^{\lambda_i \tau} \varphi_i(z, 0) \| + \max_{i=2,3,4} \left| e^{\lambda_i \tau} - e^{\lambda_i \tilde{\tau}} \right| \cdot \frac{1}{9}. \]

The two terms on the right-hand side are estimated in the following two lemmas.
Lemma 16. \(|\varphi_i(z, \tau) - \varphi_i(z, 0)e^{\lambda_i \tau}| \leq 8.72 \times 10^{-10} e^{M\lambda_4 \tilde{\tau}}\) for \(i = 2, 3\) and \(4\).

Proof. In the differential equation \(\dot{\varphi_i}(z, t) = \lambda_i \varphi_i(z, t) + G_i(\varphi(z, t))\) we use that \(|G_i(\varphi(z, t))| \leq \kappa(1 + \kappa_3)^M \varphi_1(z, 0)^M e^{M\lambda_4 t}(1 + \kappa_2)^M \varphi_4(z, 0)^M e^{M\lambda_4 t}\) according to Lemma 14 and 15. This shows that

\[
\dot{\varphi_i}(z, t) - \lambda_i \varphi_i(z, t) \leq \kappa(1 + \kappa_2)^M(1 + \kappa_3)^M \varphi_1(z, 0)^M \varphi_4(z, 0)^M e^{M(\lambda_1 + \lambda_4) t}.
\]

Now apply Grönwall’s lemma to this differential inequality,

\[
\varphi_i(z, \tau) \leq \varphi_i(z, 0)e^{\lambda_i \tau} + \kappa(1 + \kappa_2)^M(1 + \kappa_3)^M \varphi_1(z, 0)^M \varphi_4(z, 0)^M \\
\times e^{\lambda_i \tau \int_0^\tau e^{[M\lambda_1 - \lambda_i + M\lambda_4]s} ds}
\]

\[
\leq \varphi_i(z, 0)e^{\lambda_i \tau} + \frac{\kappa(1 + \kappa_2)^M(1 + \kappa_3)^M}{M\lambda_1 - \lambda_i + M\lambda_4} [\varphi_1(z, 0)e^{\lambda_1 \tau}]^M \varphi_4(z, 0)^M e^{M\lambda_4 \tilde{\tau}}.
\]

In the second term in the right-hand side we substitute \(\tau\) for \(\tilde{\tau} - (1/\lambda_1) \log(1 - \kappa_1)\) in \(e^{\lambda_i \tau}\) and \(\tau\) for \(\tilde{\tau} - (1/\lambda_1) \log(1 + \kappa_1)\) in \(e^{M\lambda_4 \tilde{\tau}}\) to get

\[
\varphi_i(z, \tau) \leq \varphi_i(z, 0)e^{\lambda_i \tau} + \frac{\kappa(1 + \kappa_1)^{-M\lambda_4/\lambda_1}(1 + \kappa_2)^M(1 + \kappa_3)^M(1/\lambda_1)^M r^M}{(M\lambda_1 - \lambda_i + M\lambda_4)(1 - \kappa_1)^M} e^{M\lambda_4 \tilde{\tau}}.
\]

By working with \(-\varphi_i(z, t)\) instead and adjusting the estimates appropriately we can derive the lower bound. \(\Box\)

Lemma 17. \(|e^{\lambda_i \tau} - e^{\lambda_i \tilde{\tau}}| \leq 2.46 \times 10^{-9} e^{\lambda_i \tilde{\tau}}\) for \(i = 2, 3\) and \(4\).

Proof. By using the exit time estimate (*) we get that

\[
e^{\lambda_i \tau} - e^{\lambda_i \tilde{\tau}} \leq e^{\lambda_i \tilde{\tau}}[(1 + \kappa_1)^{-\lambda_i/\lambda_1} - 1] \leq 2.46 \times 10^{-9} e^{\lambda_i \tilde{\tau}}.
\]

and

\[
e^{\lambda_i \tau} - e^{\lambda_i \tilde{\tau}} \geq e^{\lambda_i \tilde{\tau}}[(1 - \kappa_1)^{-\lambda_i/\lambda_1} - 1] \geq -2.46 \times 10^{-9} e^{\lambda_i \tilde{\tau}}.
\]

Combining these two lemmas we get the next result.

Result 7. \(\|\varphi(z, \tau) - e^{A\tilde{\tau}}\varphi(z, 0)\|_{\infty} \leq 1.15 \times 10^{-8} e^{[\lambda_1 + M\lambda_4] \tilde{\tau}}\).

We now do the corresponding estimate for the derivatives of \(\varphi\).

Lemma 18. \(\|D\varphi(z, \tau) - e^{A\tilde{\tau}}\|_{\infty} \leq 2.70 \times 10^{-8} e^{[\lambda_1 + (M - 1)\lambda_4] \tilde{\tau}}\).

Proof. The derivatives \(D\varphi(z, \tau)\) are obtained by solving the equations of variations,

\[
\frac{d}{dt}D\varphi(z, t) = (A + DG(\varphi(z, t))) D\varphi(z, t).
\]
We collect both terms in the left-hand side and multiply (from the left) by the integrating factor. This gives us that

\[
\frac{d}{dt} \left[ \exp \left( - \int_0^t (A + DG(\varphi(z,s))) \, ds \right) D\varphi(z,t) \right] = 0.
\]

After integrating this equality we arrive at

\[
D\varphi(z,\tau) = \exp \left( \int_0^\tau (A + DG(\varphi(z,s))) \, ds \right).
\]

The estimate in the lemma will be proved by using the following perturbation inequality (see Theorem 4 in [Bit 17 (1977), pages 39–57])

\[
\|e^{A+B} - e^A\| \leq e^{\mu(A)} (e^{\|B\|} - 1),
\]

where \(\mu(A)\) is the logarithmic norm of the matrix \(A\) which in this case, since \(A\) is diagonal, is equal to \(\mu_\infty(A) = \lambda_1\). We therefore have that

\[
\|D\varphi(z,\tau) - e^{A\tau}\|_\infty \leq e^{\lambda_1 \tau} \left[ \exp \left( \int_0^\tau \|DG(\varphi(z,s))\|_\infty \, ds \right) - 1 \right]. \tag{16}
\]

From Result 6, Lemma 14 and 15 we get an estimate of \(DG\),

\[
\|DG(\varphi(z,s))\|_\infty \leq \kappa'(1 + \kappa_2)^M - 1(1 + \kappa_3)^M - 1 \varphi_1(z,0)^{M-1} \times \varphi_4(z,0)^{M-1} e^{(M-1)(\lambda_1 + \lambda_4)s}
\]

and the integral in the norm inequality (16) is bounded by

\[
\kappa'(1 + \kappa_2)^M - 1(1 + \kappa_3)^M - 1 \varphi_1(z,0)^{M-1} \varphi_4(z,0)^{M-1} \int_0^\tau e^{(M-1)(\lambda_1 + \lambda_4)s} \, ds \leq \kappa'(1 + \kappa_2)^M - 1(1 + \kappa_3)^M - 1 \left[ \varphi_1(z,0) e^{\lambda_1 \tau} \right]^{M-1} \varphi_4(z,0)^{M-1} e^{(M-1)(\lambda_1 + \lambda_4)\tilde{\tau}}.
\]

In the factor \(\varphi_1(z,0) e^{\lambda_1 \tau}\) above we replace \(\tau\) with \(\tilde{\tau} - (1/\lambda_1) \log(1 - \kappa_1)\) and in \(e^{(M-1)(\lambda_1 + \lambda_4)\tilde{\tau}}\) replace \(\tau\) with \(\tilde{\tau} - (1/\lambda_1) \log(1 + \kappa_1)\) to obtain that the integral is smaller than

\[
\frac{\kappa'(1 + \kappa_1)^{-(M-1)\lambda_4/\lambda_1} (1 + \kappa_2)^M - 1(1 + \kappa_3)^M - 1 \left( \frac{1}{\lambda_1} \right)^{M-1} r^{M-1} e^{(M-1)\lambda_4 \tilde{\tau}}}{(M-1)(\lambda_1 + \lambda_4)(1 - \kappa_1)^{M-1}}.
\]

where the constant in front of \(e^{(M-1)\lambda_4 \tilde{\tau}}\) is smaller than \(2.69 \times 10^{-8}\). If we now use this estimate in the norm inequality (16) and use that \(e^{\tilde{\tau}} - 1 \leq \frac{x}{(1 - c)}\) for \(x \leq c \leq 1\), we get that

\[
\|D\varphi(z,\tau) - e^{A\tau}\|_\infty \leq \frac{2.69 \times 10^{-8}}{1 + \kappa_1} e^{[\lambda_1 + (M-1)\lambda_4] \tilde{\tau}} \leq 2.70 \times 10^{-8} e^{[\lambda_1 + (M-1)\lambda_4] \tilde{\tau}}. \quad \square
\]
Lemma 19. \[ |e^{\lambda_1 \tau} - e^{\lambda_1 \bar{\tau}}| \leq 1.32 \times 10^{-9} e^{[\lambda_1 + M \lambda_4] \bar{\tau}}. \]

Proof. From the inequality in Lemma 15 we have that
\[ |\varphi_1(z, t) - \varphi_1(z, 0) e^{\lambda_1 t}| \leq \kappa_3 \varphi_1(z, 0) e^{(\lambda_1 + M \lambda_4) t}. \]
If we put \( t = \tau \) and use the exit time estimate (\( \star \)) on the right-hand side we get
\[ |z - \varphi_1(z, 0) e^{\lambda_1 \tau}| \leq \kappa_3 (1 + \kappa_1) \frac{\lambda_1 + M \lambda_4}{\lambda_1} \varphi_1(z, 0) e^{(\lambda_1 + M \lambda_4) \bar{\tau}}. \]
Using that \( \varphi_1(z, 0)/r = e^{\lambda_1 \bar{\tau}} \) in this inequality shows that
\[ |e^{\lambda_1 \bar{\tau}} - e^{\lambda_1 \tau}| \leq \kappa_3 (1 + \kappa_1) \frac{\lambda_1 + M \lambda_4}{\lambda_1} e^{[\lambda_1 + M \lambda_4] \bar{\tau}}. \]

Once these lemmas are done it is a simple task to derive the main estimate
\[ \|D\varphi(z, \tau) - e^{A\bar{\tau}}\|_\infty \leq \|D\varphi(z, \tau) - e^{A\tau}\|_\infty + \max_i |e^{\lambda_i \tau} - e^{\lambda_i \bar{\tau}}| \]
\[ \leq 2.70 \times 10^{-8} e^{[\lambda_1 + (M - 1) \lambda_4] \bar{\tau}} + 2.46 \times 10^{-9} e^{[\lambda_1 + M \lambda_4] \bar{\tau}} \]
\[ \leq 2.95 \times 10^{-8} e^{[\lambda_1 + (M - 1) \lambda_4] \bar{\tau}}. \]

Result 8. \( \|D\varphi(z, \tau) - e^{A\bar{\tau}}\|_\infty \leq 2.96 \times 10^{-8} e^{[\lambda_1 + (M - 1) \lambda_4] \bar{\tau}}. \)

The results so far show that the flow variables of the normal form system and the linear system differ very little (when \( M = 6 \) is chosen so that \( \lambda_1 + (M - 1) \lambda_4 < 0 \)). Our last result will use this fact to also show that when comparing the differential of the mapping from the top of the cube onto the exit faces \( z_1 = \pm r = \pm \frac{1}{10} \) there is a relatively small discrepancy between using the the normal form system and the linear system. This result is used in Section 2.8 and 2.9 in Chapter 2 to correctly estimate aperture and expansion inside cones as they flow through the cube. In the proof of this last result we will use the following two elementary lemmas (given here without proof).

Lemma 20. If \( z \in \{ z : z_1 = r; \ |z_2|, |z_3| \leq 10^{-5}; \ 0 < z_4 \leq r \} \) then \( |f_1(z)| > 1.2 \) and \( |f_i(z)/f_1(z)| \leq 0.3 \) for \( i = 2, 3 \) and 4.

Lemma 21. If \( z \in \{ z : z_1 = r; \ |z_2|, |z_3| \leq 2 \times 10^{-5}; \ 0 < z_4 \leq r + 10^{-5} \} \) then \( \|Df_1(z)\|_\infty \leq 12, \|Df_2(z)\|_\infty \leq 12, \|Df_3(z)\|_\infty \leq 22 \) and \( \|Df_4(z)\|_\infty \leq 4. \)

Let \( L(z) = \varphi(z, \tau(z)) \) be the mapping through the cube with the normal form system and let \( \bar{L}(z) = \varphi(z, 0) e^{A\bar{\tau}(z)} \) be the same mapping but with the linear system.

Result 9. If \( \lambda_1 + (M - 1) \lambda_4 < 0 \) and \( z \in \{ z : 0 < z_1 \leq r; \ |z_2|, |z_3| \leq 10^{-5}; \ z_4 = r \} \) then \( \|DL(z) - D\bar{L}(z)\|_\infty \leq 6.52 \times 10^{-7} (z_1/r)^{-2 - M \lambda_4/\lambda_1}. \)
Remark. The exponent $-2 - M\lambda_4/\lambda_1$ does not need to be positive for our purposes.

Proof. The derivatives of $L(z)$ are calculated with the formula

$$
\frac{\partial L_i}{\partial z_j}(z) = \frac{\partial \varphi_i}{\partial z_j}(z, \tau) - \frac{f_i(\varphi(z, \tau))}{f_1(\varphi(z, \tau))} \frac{\partial \varphi_1}{\partial z_j}(z, \tau).
$$

(See the derivation of the derivative of the return map in Section 2.2 in Chapter 2.)

In matrix form this formula can be written as

$$
DL(z) = D\varphi(z, \tau) - \frac{f(\varphi(z, \tau))}{f_1(\varphi(z, \tau))} D\varphi_1(z, \tau),
$$

where $D\varphi = (D\varphi_2, D\varphi_3, D\varphi_4)$ and $f = (f_2, f_3, f_4)$. By letting $\bar{\varphi}$ denote the flow of the linear system we have that $\|DL(z) - DL(\bar{z})\|_\infty$ is bounded by

$$
\|D\varphi(z, \tau) - D\bar{\varphi}(z, \bar{\tau})\|_\infty + \frac{\|f(\varphi(z, \tau))\|}{f_1(\varphi(z, \tau))} \|D\varphi_1(z, \tau) - D\bar{\varphi}_1(z, \bar{\tau})\|_\infty.
$$

The leftmost term can be estimated using Result 8 while for the rightmost term we use the inequality

$$
\|(A + E_1)(B + E_2) - AB\| \leq \|A\|\|E_2\| + \|B\|\|E_1\| + \|E_1\|\|E_2\|
$$

and get that it is bounded by

$$
\left\| \frac{f(\bar{\varphi}(z, \bar{\tau}))}{f_1(\bar{\varphi}(z, \bar{\tau}))} \right\| \|D\varphi_1(z, \tau) - D\bar{\varphi}_1(z, \bar{\tau})\|_\infty
$$

$$
+ \|D\bar{\varphi}_1(z, \bar{\tau})\|_\infty \left\| \frac{f(\varphi(z, \tau))}{f_1(\varphi(z, \tau))} - \frac{f(\bar{\varphi}(z, \bar{\tau}))}{f_1(\bar{\varphi}(z, \bar{\tau}))} \right\|_\infty
$$

$$
+ \|D\varphi_1(z, \tau) - D\bar{\varphi}_1(z, \bar{\tau})\|_\infty \left\| \frac{f(\varphi(z, \tau))}{f_1(\varphi(z, \tau))} - \frac{f(\bar{\varphi}(z, \bar{\tau}))}{f_1(\bar{\varphi}(z, \bar{\tau}))} \right\|_\infty.
$$

From Lemma 20 we have that $\|f(\bar{\varphi}(z, \bar{\tau}))/f_1(\bar{\varphi}(z, \bar{\tau}))\|_\infty \leq 0.3$, the next factor $\|D\varphi_1(z, \tau) - D\bar{\varphi}_1(z, \bar{\tau})\|_\infty$ is estimated using Result 8 and $\|D\bar{\varphi}_1(z, \bar{\tau})\|_\infty = e^{\lambda_1 \bar{\tau}}$. To handle the last factor we use the following inequality

$$
\left| \frac{x + \Delta x}{y + \Delta y} - \frac{x}{y} \right| \leq \frac{\|\Delta x\|}{\|y\|} - \frac{\|\Delta y\|}{\|y\|} + \frac{\|\Delta y\|}{\|y\|} - \frac{\|\Delta y\|}{\|y\|},
$$

to conclude that

$$
\left\| \frac{f(\varphi(z, \tau))}{f_1(\varphi(z, \tau))} - \frac{f(\bar{\varphi}(z, \bar{\tau}))}{f_1(\bar{\varphi}(z, \bar{\tau}))} \right\|_\infty
$$

$$
\leq \frac{\|f(\varphi(z, \tau)) - f(\bar{\varphi}(z, \bar{\tau}))\|_\infty}{\|f_1(\varphi(z, \tau)) - f_1(\bar{\varphi}(z, \bar{\tau}))\|} + \left\| \frac{f(\bar{\varphi}(z, \bar{\tau}))}{f_1(\bar{\varphi}(z, \bar{\tau}))} \right\|_\infty \frac{\|f_1(\varphi(z, \tau)) - f_1(\bar{\varphi}(z, \bar{\tau}))\|}{\|f_1(\varphi(z, \tau)) - f_1(\bar{\varphi}(z, \bar{\tau}))\|},
$$

where $\lambda_1$ is the largest Lyapunov exponent of the linear system.
where these quantities may be estimated using Result 7, Lemma 20 and 21 as follows,

- \( \| f(\varphi(z, \tau)) - f(\tilde{\varphi}(z, \tilde{\tau})) \|_\infty \leq \| Df(\xi) \|_\infty \| \varphi(z, \tau) - \tilde{\varphi}(z, \tilde{\tau}) \|_\infty \),
- \( |f_1(\varphi(z, \tau)) - f_1(\tilde{\varphi}(z, \tilde{\tau}))| \leq \| Df_1(\xi) \|_\infty \| \varphi(z, \tau) - \tilde{\varphi}(z, \tilde{\tau}) \|_\infty \),
- \( |f_1(\tilde{\varphi}(z, \tilde{\tau}))| \geq 1.1 \), and
- \( \| f(\tilde{\varphi}(z, \tilde{\tau}))/f_1(\tilde{\varphi}(z, \tilde{\tau})) \|_\infty \leq 0.3 \).

These estimates put together show that

\[
\left\| \frac{f(\varphi(z, \tau))}{f_1(\varphi(z, \tau))} - \frac{f(\tilde{\varphi}(z, \tilde{\tau}))}{f_1(\tilde{\varphi}(z, \tilde{\tau}))} \right\|_\infty \leq 2.18 \times 10^{-8} e^{[\lambda_1 + M\lambda_4] \tilde{\tau}}
\]

and the whole expression \( \| DL(z) - D\tilde{L}(z) \|_\infty \) is bounded by

\[
1.3 \cdot 2.95 \times 10^{-8} e^{[\lambda_1 + (M-1)\lambda_4] \tilde{\tau}} + 2.18 \times 10^{-8} e^{[2\lambda_1 + M\lambda_4] \tilde{\tau}} + 2.95 \times 10^{-8} \cdot 2.18 \times 10^{-8} e^{[\lambda_1 + (M-1)\lambda_4] \tilde{\tau}}.
\]

As \( e^{\lambda_1 \tilde{\tau}} = r/z_1 \) and \( 0 < z_1 \leq r \) this shows that

\[
\| DL(z) - D\tilde{L}(z) \|_\infty \leq (3.84 \times 10^{-8} + 2.18 \times 10^{-8}) \left( \frac{z_1}{r} \right)^{-2 - M\lambda_4/\lambda_1}.
\]

### 3.6 The Lorenz system

The Lorenz system

\[
\begin{aligned}
\dot{x}_1 &= \lambda_1 x_1 - k_1(x_1 + x_2)x_3 \\
\dot{x}_2 &= \lambda_2 x_2 + k_1(x_1 + x_2)x_3 \\
\dot{x}_3 &= \lambda_3 x_3 + (x_1 + x_2)(k_2 x_1 + k_3 x_3)
\end{aligned}
\]

is transformed to a normal form system \( \dot{z} = Az + G(z) \), where \( A = \text{diag}(\lambda_1, \lambda_2, \lambda_3) \) and \( |G_i(z)| \leq C |z_1|^M (|z_2| + |z_3|)^M \) for \( M = 6 \), in practically the same way as the Lorenz-Stenflo system. The calculations in Section 3.2 to 3.5 can therefore more or less be repeated verbatim for the Lorenz system, and in the rest of this section we only provide a summary of the results one obtains for this system.

#### The coordinate change function \( \phi \)

The first coordinate change function \( x = \phi(z) \) is written as a power series \( \phi_i(z) = \sum_j c_{i,j} z^j \) where the coefficients \( c_{i,j} \) satisfy the recurrence equations
3.6. **THE LORENZ SYSTEM**

![Graph](image)

**Figure 3.3:** A plot of $\Omega_t(j) = \min_{j \in J_t} |\Omega_t(j)|$ which grows linearly with $j$ after an initial transient part.

- $\Omega_1(j)c_{1,j} = -k_1 \sum_m (c_{1,m} + c_{2,m})c_{3,j-m}$,

- $\Omega_2(j)c_{2,j} = k_1 \sum_m (c_{1,m} + c_{2,m})c_{3,j-m}$,

- $\Omega_3(j)c_{3,j} = \sum_m (c_{1,m} + c_{2,m})(k_2c_{1,j-m} + k_3c_{2,j-m})$,

for $j$ such that $j_1 \leq M - 1$ or $j_2 + j_3 \leq M - 1$, and otherwise $c_{i,j} = 0$. In the recurrence equations $\Omega_t(j) = j_1\lambda_1 + j_2\lambda_2 + j_3\lambda_3 - \lambda_i$ is an integer linear combination of the values $\lambda_1$, $\lambda_2$ and $\lambda_3$, and plays the same pivotal role as in the Lorenz-Stenflo system. We calculate $\phi$ by truncating its power series to degree $N = 8$ and estimate the remainder term by introducing a sequence $\{h_{i,j}\}_j$ such that the conditions $h_{i,j} \geq \sum_{|j|=j} |c_{i,j}|$ hold and these numbers satisfy a set of simpler recurrence equations:

- $h_{1,j} = \frac{|k_1|}{\Omega_1(j)} \sum_{m=1}^{j-1} (h_{1,m} + h_{2,m})h_{3,j-m}$,

- $h_{2,j} = \frac{\Omega_1(j)}{\Omega_2(j)} h_{1,j}$, and

- $h_{3,j} = \frac{1}{\Omega_3(j)} \sum_{m=1}^{j-1} (h_{1,m} + h_{2,m})(|k_2|h_{1,j-m} + |k_3|h_{2,j-m})$,

where $\Omega_t(j) = \min_{j \in J_t} |\Omega_t(j)|$ and $J_t = \{j : |j| = j$ and $(j_1 < M$ or $j_2 + j_3 < M)\}$. For index $j > N' = 40$ we prove by induction that

$$h_{1,j} \leq 3.75 \times 10^{-13} \cdot 5^j, \quad h_{2,j} \leq 1.48 \times 10^{-12} \cdot 5^j, \quad \text{and} \quad h_{3,j} \leq 2.54 \times 10^{-12} \cdot 5^j.$$ Combining these results shows that $\|\phi(z) - \phi^T(z)\|_\infty \leq 1.23 \times 10^{-14}$ for $\|z\|_\infty \leq \frac{1}{10}$ and $\|\phi(z) - \phi^T(z)\|_\infty \leq 5.60 \times 10^{-14}$ for $\|z\|_\infty \leq \frac{1}{10}$, where $\phi^T(z)$ is the truncated power series. A similar calculation also shows that $\|D\phi(z) - D\phi^T(z)\|_\infty \leq 1.75 \times 10^{-12}$ for $\|z\|_\infty \leq \frac{1}{10}$.
The coordinate change function $\psi$

The opposite coordinate change function $z = \psi(x)$ is also written as a power series $\psi_i(x) = \sum_j e_{i,j}x^j$ and the coefficients $e_{i,j}$ satisfy the recurrence equation

$$\Omega_i(j)e_{i,j} = e_{i,j-e_3,j_1}k_1 + e_{i,j+e_1-e_2-e_3}(j_1 + 1)k_1 - e_{i,j-e_1+e_2-e_3}(j_2 + 1)k_1$$
$$- e_{i,j-e_3,j_2}k_1 - e_{i,j-2e_1+e_3}(j_3 + 1)k_2 - e_{i,j-2e_2+e_3}(j_3 + 1)k_3$$
$$- e_{i,j-e_1+e_2+e_3}(j_3 + 1)(k_2 + k_3)$$

assuming that $|j| \leq 2M$. This power series of $\psi$ is truncated to degree $N$ and the remainder term is shown to be small by first proving that

$$\|\psi^T \circ \phi^T(z) - z\|_\infty \leq 1.50 \times 10^{-13} \quad \text{and} \quad \|D(\psi^T \circ \phi^T)(z) - I\|_\infty \leq 1.56 \times 10^{-11},$$

for $\|z\|_\infty \leq \frac{1}{5}$, using the same type of calculations as in Lemma 5, and then performing the rest of the calculations in Section 3.3. The end result is that we manage to prove that

$$\|\psi(x) - \psi^T(x)\|_\infty \leq 2.10 \times 10^{-13} \quad \text{and} \quad \|D\psi(x) - D\psi^T(x)\|_\infty \leq 1.56 \times 10^{-11}$$

for $\|x\|_\infty \leq \frac{1}{17}$.

The non-linear term $G$

Turning to the non-linear term $G(z)$ its power series $G_i(z) = \sum_j d_{i,j}z^j$ has coefficients $d_{i,j}$ that satisfy the following recurrence equations

- $d_{1,j} = k_1 \sum_m (c_{1,m} + c_{2,m})c_{3,j-m}$
  $$- \sum_{|m| \geq 1} (c_{1,m+e_1}(m_1 + 1)d_{1,j-m} + c_{1,m+e_2}(m_2 + 1)d_{2,j-m}$$
  $$+ c_{1,m+e_3}(m_3 + 1)d_{3,j-m},$$

- $d_{1,j} = -k_1 \sum_m (c_{1,m} + c_{2,m})c_{3,j-m}$
  $$- \sum_{|m| \geq 1} (c_{2,m+e_1}(m_1 + 1)d_{1,j-m} + c_{2,m+e_2}(m_2 + 1)d_{2,j-m}$$
  $$+ c_{2,m+e_3}(m_3 + 1)d_{3,j-m},$$

- $d_{1,j} = -\sum_m (c_{1,m} + c_{2,m})(k_{2c_{1,j-m}} + k_3c_{2,j-m})$
  $$- \sum_{|m| \geq 1} (c_{3,m+e_1}(m_1 + 1)d_{1,j-m} + c_{3,m+e_2}(m_2 + 1)d_{2,j-m}$$
  $$+ c_{3,m+e_3}(m_3 + 1)d_{3,j-m},$$
for \( j \) such that \( j_1 \geq M \) and \( j_2 + j_3 \geq M \), and otherwise \( d_{i,j} = 0 \). In order to estimate \( G \) we introduce a sequence \( \{ g_{i,j} \}_j \) that majorizes the sums \( \sum_{|j|=j} |d_{i,j}| \) and can be calculated with a scaled-down version of the recurrence equations for \( d_{i,j} \),

\[
\begin{align*}
g_{1,j} &= |k_1| \sum_{m=1}^{j-1} (h_{1,m} + h_{2,m}) h_{3,j-m} \\
&\quad + \sum_{m=1}^{j-2M} (m+1) h_{1,m+1} (g_{1,j-m} + g_{2,j-m} + g_{3,j-m}), \\
g_{2,j} &= |k_1| \sum_{m=1}^{j-1} (h_{1,m} + h_{2,m}) h_{3,j-m} \\
&\quad + \sum_{m=1}^{j-2M} (m+1) h_{2,m+1} (g_{1,j-m} + g_{2,j-m} + g_{3,j-m}), \\
g_{3,j} &= |k_1| \sum_{m=1}^{j-1} (h_{1,m} + h_{2,m}) (|k_2| h_{1,j-m} + |k_3| h_{2,j-m}) \\
&\quad + \sum_{m=1}^{j-2M} (m+1) h_{3,m+1} (g_{1,j-m} + g_{2,j-m} + g_{3,j-m}),
\end{align*}
\]

for \( j > P \) and \( g_{i,j} = \sum_{|j|=j} |d_{i,j}| \) for \( j \leq P \). By induction we can prove that

\[
g_{1,j} \leq 6.59 \times 10^{-15} \cdot 7^j, \quad g_{2,j} \leq 6.63 \times 10^{-15} \cdot 7^j \quad \text{and} \quad g_{3,j} \leq 2.57 \times 10^{-14} \cdot 7^j
\]

for \( j > N' \). With these results we can factor out common subexpressions and show that \( \|G(z)\|_\infty \leq \kappa |z_1|^M (\max\{|z_2|, |z_3|\})^M \) if \( \|z\|_\infty \leq \frac{1}{5} \) and where \( \kappa = 8.55 \times 10^{-3} \) as well as \( \|DG(z)\|_\infty \leq \kappa' |z_1|^{M-1} (\max\{|z_2|, |z_3|\})^{M-1} \) where \( \kappa' = 1.45 \times 10^{-2} \).

**The normal form flow**

A comparative analysis of the normal form flow and the flow of the linear system \( \dot{z} = Az \) inside the cube \( \|z\|_\infty \leq \frac{1}{10} \), as we did in Section 3.5, results in the following two estimates

\[
\begin{align*}
\|\varphi(z, \tau) - e^{A\tau} \varphi(z, 0)\|_\infty &\leq 3.32 \times 10^{-15} e^{[\lambda_1 + M\lambda_3] \bar{\tau}} \quad \text{and} \\
\|D\varphi(z, \tau) - e^{A\tau}\|_\infty &\leq 6.10 \times 10^{-14} e^{[\lambda_1 + (M-1)\lambda_3] \bar{\tau}},
\end{align*}
\]

where \( \varphi(z, t) \) is the normal form flow, \( e^{A\tau} \varphi(z, 0) \) is the linear flow, \( \tau \) and \( \bar{\tau} \) are the exit times of the normal form and linear systems, respectively.

By our choice of \( M = 6 \) we have that \( \lambda_1 + (M-1)\lambda_3 < 0 \) and then it can be shown that

\[
\|DL(z) - D\tilde{L}(z)\|_\infty \leq 1.49 \times 10^{-12} (z_1/r)^{-2 - M\lambda_3/\lambda_1},
\]

where \( L \) and \( \tilde{L} \) are the mapping through the cube with the normal form and linear systems, respectively.
Chapter 4

Results

In this final chapter the results from the computer runs are summarised and the practical details of running the programs are explained.

The programs used in this thesis are downloadable from the web page

http://www.math.kth.se/~ekola/thesis/programs.tgz

This file unpacks into two directories lorenz/ and lorenz-stenflo/ which contain files that correspond to the Lorenz and Lorenz-Stenflo systems. Each of these directories has the following subdirectories:

- m-files/ includes the Matlab functions used by the main programs step01.m to step15.m.
- phipsi/ is where the Maple programs discussed in Chapter 3 are stored.
- result/ keeps a copy of all result files generated by the main programs step01.m to step15.m.
- step00/, step01-02/,...,step14-15/ are the directories where the main programs are stored.

Below is a step by step account of how to run the main programs and also some of the more important results.

Step 0

Before the Matlab runs can start a few Matlab functions need to be generated by the five Maple programs: lsodedata.mpl, box_R.mpl, R.mpl, xphi.mpl and xpsi.mpl. A typical way to run these programs is to start Maple and type in the following lines:

```matlab
restart; read "lsodedata.mpl":
restart; read "box_R.mpl":
```
restart; read "R.mpl":
restart; read "xphi.mpl":
restart; read "xpsi.mpl":

After executing these lines the Matlab functions lsodedata.m, lsdata.m, box_R.m, R.m, xphi.m and xpsi.m are output in the directory m-files/.

Step 1 and 2

The first two steps are run in the following way:

1. Change to the directory step01-02/ and type ‘make’ at the shell prompt. This will clean up the directory.

2. Start Matlab and type ‘step01’ followed by ‘step02’.

The output is stored in the subdirectory result/ and in the files unfinished.dat (the box created in step01.m) and orbit.mat (the points from step02.m), plus the empty data files variables.mat and added_boxes.mat.

Step 3 to 6

Step 3 to 6 is a combination of steps that creates the collection of invariant boxes. In a first phase, step 3 and 4 produce a set of boxes that form an almost invariant collection of boxes. This set is in step 5 and 6 augmented with additional boxes after a more detailed study of the boxes that flow close to the origin. By reapplying step 3 and 4 to the added boxes, a full set of invariant boxes is created.

Run the programs step03.m and step04.m by doing the following steps:

1. Run ‘make from01-02’ in the directory step03-04/ to copy the output files from step 1 and 2 to the current directory.

2. Start Matlab and type ‘step03’. Several copies of step03.m can be run in parallel and each process is launched in this way.

3. Start Matlab and type ‘step04’. The program step04.m oversees all instances of step03.m and once they finish, it outputs the resulting box covering in the files unfinished.dat (a list of boxes prepared for step 5 and 6) and variables.mat (all boxes are listed in this file) in the subdirectory result/.

Depending on how many copies of step03.m are running and the speed of the computers it normally takes several hours to finish these two steps.

In step 5 and 6 the mappings from boxes crossing the discontinuity of the return map are analysed and programs step05.m and step06.m are run in the same way as step03.m and step04.m:

1. Do ‘make from03-04’ in the directory step05-06/ to copy the output files from step 3 and 4.
Figure 4.1: The leftmost diagram is a plot of the box covering of the Lorenz attractor while the rightmost diagram is the box covering of the Lorenz-Stenflo attractor. The coverings are comprised of 1658 boxes and 1894 boxes, respectively, on each branch of the attractor.

2. Start Matlab and type ‘step05’. This program can be run as many parallel processes, but running more instances than there are discontinuity boxes has no effect.

3. Start Matlab and type ‘step06’.

Once step06 has finished it outputs a text explaining if any new boxes were added to the covering or not. If new boxes are added (which they are for both the Lorenz and Lorenz-Stenflo systems) then it is necessary to rerun step03.m and step04.m before continuing to step 7:

1. Change to directory step03-04/ and run ‘make from05-06’. The output files from step05-06/ are copied to the current directory.

2. Run step03.m and step04.m as before. This time it is probably enough to run step03.m as a single process.

Figure 4.1 is a plot of the box covering for the Lorenz and Lorenz-Stenflo attractors.

Step 7 to 11
The next steps produce invariant cones on all the boxes of the collection in the following way:

- step07.m creates an initial set of cones on just a few boxes. These cones contain directions that are certain to be inside any set of invariant cones.

- step08.m and step09.m use an iterative method of pushing cones forward to create cones that are invariant under a piecewise constant approximation of the differential map of the return map.
- step10.m and step11.m take the cones generated by step08.m and step09.m and use the same type of iterative method to create invariant cones.

Step 7 is run in the standard way:

1. In directory step07/ do `make from03-04`. If the last steps had been step 5 and 6 then `make from05-06` should be used.

2. Start Matlab and execute `step07`. The created cones are output in the file result/cones.mat.

Step 8 and 9 should be run alongside each other similar to step 3 and 4:

1. Do first `make` in the directory step08-09/.

2. Start step08.m in Matlab. The program step08.m is capable of being run as several parallel processes but since it handles each box fairly fast it usually suffice to run it as one process.

3. Start step09.m in Matlab. The main task of step09.m is to manage the iterations that are involved in the algorithm.

The creation of the invariant cones in step 10 and 11 is done in the following way:

1. `make` in the directory step10-11/.

2. Start step11.m in Matlab. Step 11 is started before step 10 and this is because it flows cones down to the cube centred at the origin before step 10 can take over.

3. Start multiple instances of step10.m in Matlab.

Step 10 and 11 will take a few iterations before a set of invariant cones emerge. The output cones are stored in the file result/variables.mat.

Figure 4.2 shows how the aperture of the cones varies over the boxes.

**Step 12 to 15**

In step 12 and 13 each mapping from one box to another box is studied and the smallest expansion factor is calculated when a vector inside the outer cone of the first box is mapped to the inner cone of the second box. These expansion factors are put together in a graph that models the return map (see Section 2.10) and in step 14 and 15 this graph is analysed to determine if all periodic orbits (cycles in the graph) have expansion greater than 1 and if mappings between two boxes that cross the discontinuity of the return map have expansion greater than 2.

The programs step12.m and step13.m are run in the same way as step10.m and step11.m:

1. Type `make` in the directory step12-13/.
Figure 4.2: These diagrams show the apertures of the outer and inner cones on each box (in black and grey colour, respectively). The aperture of a cone is measured as the largest side length of the rectangle in the perpendicular plane to the central direction of the cone. The top diagram is for the Lorenz system and bottom one is for the Lorenz-Stenflo system.

2. Start step13.m in Matlab. Besides managing step12.m this program calculates expansions down to the cube around the origin and therefore needs to be started first.

3. Start multiple instances of step12.m in Matlab.

After the programs have finished, which could take several days, the expansion data is stored in the file `result/box_graph.dat` as well as together with other variables in the file `result/variables.mat`.

The two remaining programs `step14.m` and `step15.m` depend on external graph analysing programs that first need to be installed manually. These programs are downloaded from the web page


and compiled. The paths to the programs `bfct` and `gor1` are then added to the lines `path_to_bfct='...'` and `path_to_gor1='...'` in `step14.m` and `step15.m`, respectively. The two steps are run in the following way:

1. Run ‘make’ in the directory `step14-15/`.

2. Start `step14.m` in Matlab. The program `step14.m` checks if the graph has any negative cycles (periodic orbits with expansion less than 1) by invoking the
program `bfct`. It redirects the output of `bfct` to the file `result/negative_cycle.dat` which needs to be inspected.

3. Start `step15.m` in Matlab. This program uses the program `gor1` to find all shortest paths in the graph and also checks if the paths between the boxes that contains the discontinuity have expansion greater than 2. A short summary of the results is output in the file `result/short_paths.dat`.

The conclusion of these last two computer runs is that the return map for both the Lorenz and Lorenz-Stenflo systems does not have any periodic orbits with expansion less than 1 and neither system has any orbits between two discontinuity boxes with expansion less than 2 (the smallest expansion is 226 and 164 for the Lorenz and Lorenz-Stenflo systems, respectively). This numerical study therefore supports the conclusion that the Lorenz and Lorenz-Stenflo systems have strange attractors.
Appendix A

References

This appendix collects and gives more detailed data of the references used in the various sections, to complement the short form used there. In certain cases additional references are included when this is deemed appropriate.

Preface


Section 1.1


Section 1.2


Section 1.3


Section 1.4


Section 1.5


Section 1.6


Other studies of the Lorenz-Stenflo system:


**Section 2.10**


**Section 3.3**


**Section 3.5**
