A Computer Assisted Proof of Universality for for Cubic Critical Maps of the Circle with Golden Mean Rotation Number

by

Benjamin David Mestel

Thesis submitted for the degree of Doctor of Philosophy to the University of Warwick

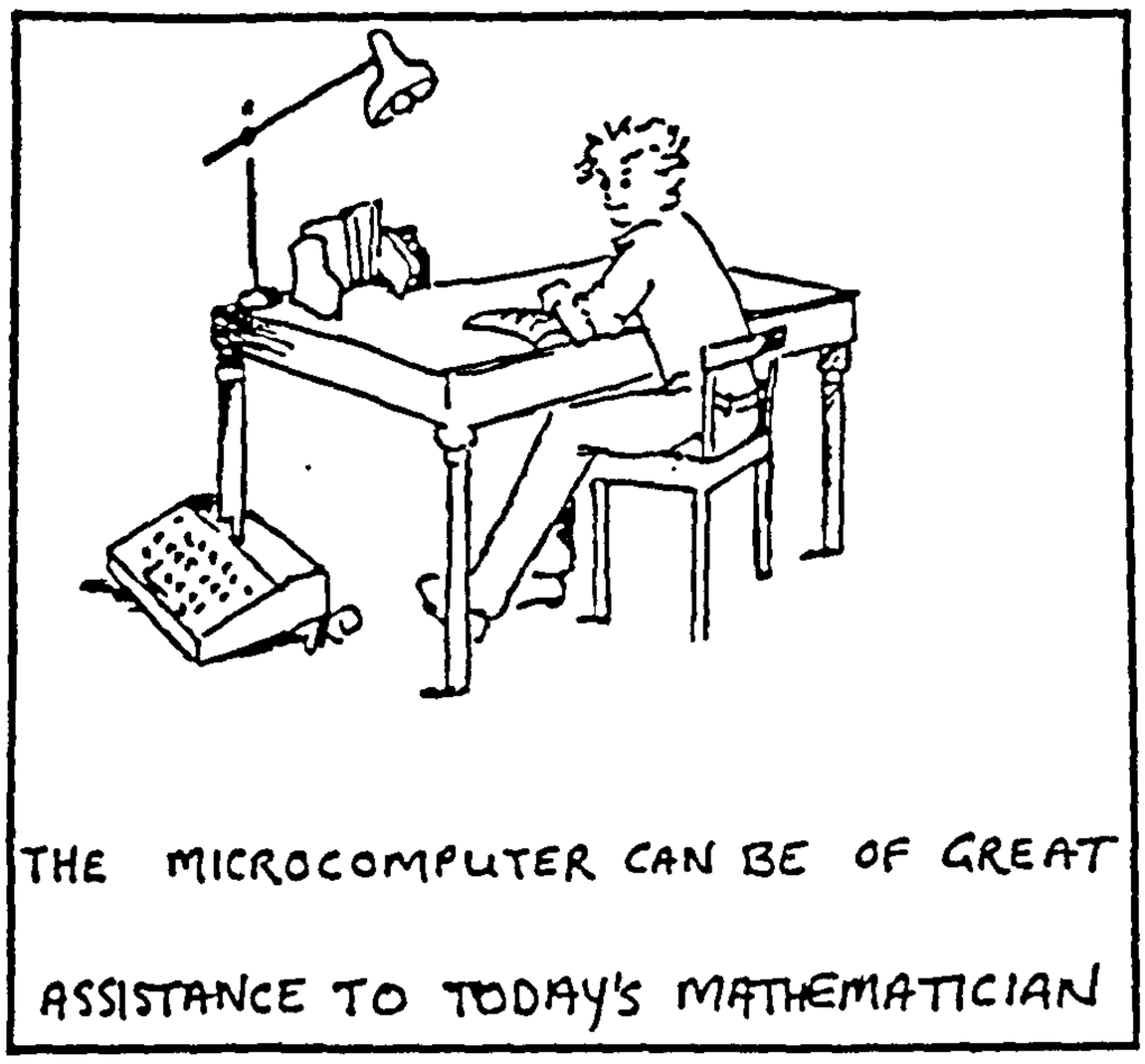
Mathematics Institute

University of Warwick

Coventry CV4 7AL

United Kingdom

September 1985



An illustration from Micro Maths by Keith Devlin, a new paperback based on the author's Guardian columns, to be published by Macmillan Educational on December 13.

Acknowledgements

I am indebted to my supervisor David Rand for the enormous amount of help and encouragement he has given throughout my time at Warwick. Thanks also to Martin Casdagli and to Robert McKay. I should like generally to thank the staff and students of the Mathematics Institute of the University of Warwick for making my time there so enjoyable. Special thanks must go to Professor Zeeman and to Elaine Shiels. Many Thanks also to Phillip Holmes and the Center for Applied Mathematics at Cornell University for a stimulating and productive stay.

This research was supported by a U.K. Science and Engineering Research Council Studentship and a grant from the Cornell University Center for Applied Mathematics.

Summary

In order to explain the universal metric properties associated with the breakdown of invariant tori in dissipative dynamical systems, Ostlund, Rand, Sethna and Siggia together with Feigenbaum, Kadanoff and Shenker have developed a renormalisation group analysis for pairs of analytic functions that glue together to make a map of the circle. Using a method of Lanford's, we have obtained a proof of the existence and hyperbolicity of a non-trivial fixed point of the renormalisation transformation for rotation number equal to the golden mean ($\sqrt{5} - 1)/2$. The proof uses numerical estimates obtained rigorously with the aid of a computer. These computer calculations were based on a method of Eckmann, Koch and Wittwer.

Contents

Chapter		Page
0.	Introduction	1
1.	Physical motivation	4
2.	Renormalisation of circle maps	9
	2.1 Parametrised families of maps of the circle	9
	2.2 Scaling laws for circle maps	10
	2.3 Pairs of maps	13
	2.4 Renormalisation transformation	14
	2.5 Simple or weak coupling fixed point	17
	2.6 Critical or strong coupling fixed point	18
	2.7 Scaling laws for cubic critical maps	20
	2.8 Extensions to higher dimensions	21
	2.9 Experimental verification	22
3.	Statement and proof of results	25
	3.1 Statement of results	25
	3.2 Description of the method of proof	30
	3.3 Proof of Theorem 3.1	31
4.	Interval arithmetic and numerical functional analysis	42
	4.1 Interval arithmetic	42
	4.2 Computer implementation of interval arithmetic	44
	4.3 Numerical functional analysis	45
	4.4 Summary	49

Chapter Page

5. Philosophical and mathematical considerations..... 51

5.1 The controversy over the four colour theorem..... 52

5.2 Are these proofs "good" or "bad" mathematics?..... 56

6. References

Appendix Page

A0. Notation..... 64

A1. Continued fractions..... 67

A1.1 Simple continued fractions..... 67

A1.2 Eventually periodic continued fractions..... 67

A1.3 The golden mean σ..... 69

A2. Maps of the circle..... 70

A3. Spaces of analytic functions, analyticity improving maps, compact operators, perturbation theory and L^1-spaces..... 74

A3.1 Banach spaces of analytic functions..... 74

A3.2 Properties of the spaces L(a, r) and A(a, r)..... 76

A3.3 Analyticity improving maps..... 77

A3.4 Spectral theory and perturbation theory..... 80

A3.5 L^1-spaces..... 83

A4. Normalisation conditions and the spectrum of dT_*..... 86

A4.1 Change in normalisation condition..... 86

A4.2 Properties of (ε_*, η_*)..... 95

A4.3 Eigenvalues of dC_*..... 98

A5. Stable and unstable manifolds..... 103

Appendix Page

A6. Various formulae involving T and dT..... 105

A6.1 Formulae in terms of (ξ, η)..... 105

A6.2 Formulae in terms of (h, k)..... 107

A7. The computer programs and explanatory notes..... 110

A7.1 General notes on the computer programs..... 110

A7.2 The program circ_prep..... 115

A7.3 The program circ_proof..... 119

A8. Error bounding for VAX-11/750 computers, interval arithmetic and function ball/vector arithmetic..... 147

A8.1 Error bounding..... 147

A8.1.1 Floating point representation and the functions r_up and r_down..... 147

A8.1.2 VAX floating point arithmetic..... 150

A8.1.3 Floating point overflow..... 152

A8.2 Interval arithmetic..... 153

A8.3 Function ball/vector operations..... 163

List of Figures

Figure		Page
1.	The Hopf bifurcation.....	4
2.	An attracting 2-torus in phase space.....	5
3.	The experimental apparatus for Taylor vortex flow and Rayleigh-Bénard convection flow.....	6
4.	Spectra for Taylor vortex flow and Rayleigh-Bénard convection.....	6
5.	Photographs of Taylor vortex flow.....	6
6.	A Poincaré cross-section of an invariant 2-torus.....	7
7.	The strong stable foliation of the invariant circle of the Poincaré return map.....	7
8.	The Arnold tongues for the sine family $f_{\omega,a}$.....	10
9.	Graph of the rotation number $\rho(f_{\omega,a})$ against $2\pi\omega$ for fixed $a \in (0,1)$.....	10
10.	Scaling behaviour of Arnold tongues.....	11
11.	Constructing a circle map from a pair of functions.....	14
12.	The renormalisation transformation T for golden mean rotation number.....	15
13.	The effect of T on the rotation number of a pair (ξ, η)...	16
14.	Schematic diagram of the geometry around the fixed points of T.....	19
15.	Geometry around the cubic critical fixed point on the critical surface.....	19
16.	Power spectrum of the sine map for $a = 1$, $\rho(f_{\omega,a}) = \sigma$.....	24

Figure Page

17. Power spectrum for Rayleigh-Bénard convection..... 24

18. The critical fixed point (ξ_*, η_*)..... 28

List of Tables

Table Page

2.1 Properties of the simple fixed point..... 18

2.2 Properties of the critical fixed point..... 18

A8.1 Extract from the VAX Architecture Guide detailing D-data/double variable storage..... 147

A8.2 Extract from the VAX Architecture Guide describing the accuracy of floating point instructions..... 150

A8.3 Extract from the VAX Architecture Manual describing the floating point overflow fault..... 153

0. Introduction

In the late seventies, Feigenbaum (1978) made a remarkable discovery while studying one-parameter families of quadratic maps on an interval. Looking at the parameter values for which successive period doubling bifurcations occurred, he found that they accumulated at an asymptotically geometric rate and that this rate was universal in the sense that is was the same for many different families. Using the idea of a "renormalisation group" from the study of critical exponents in statistical physics, he developed an analysis to explain this phenomenon, which was based on a number of conjectures about a non-linear "renormalisation transformation" on function space.

This work has produced a great deal of interest in the dynamics world, not least because here were quantitative results coming from a primarily qualitative area of mathematics. The universal period doubling sequences have been found in higher dimensional dissipative systems and in real physical systems (see Collet and Eckmann (1980)). The conjectures themselves proved a tough nut to crack and were only solved by Lanford (1982) and, in part, by Campanino, Ruelle and Epstein (1981) in 1981. Both of these proofs used functional analytic estimates obtained rigorously on a digital computer.

Feigenbaum-like universal behaviour has been found in other systems undergoing a "transition to chaos." These include conservative systems where universality has been found for period doubling (Eckmann et al (1982)) and for the breakdown of K.A.M. tori (McKay (1982)). Manton and Nauenberg have discovered universal behaviour in complex dynamical systems (Manton and Nauenberg (1983), Widom (1983)).

We shall be concerned with another path to chaos found in dissipative systems, namely the famous Ruelle-Takens scenario (Ruelle and Takens (1971)):

steady state -> periodic -> quasi-periodic -> chaos

Ostlund, Rand, Sethna and Siggia (1983) and Feigenbaum, Kadanoff and Shenker (1982) have developed a renormalisation scheme to explain universal behaviour in dissipative systems undergoing the Ruelle-Takens transition. This scheme is based on one dimensional maps of the circle for which universal behaviour was discovered by Shenker (1982). This universal behaviour has now been observed experimentally (Fein et al (1985)). We have developed a Lanford-type proof of the most important conjectures made in Ostlund et al (1983). In what follows, we shall explain in more detail the background to the theory in Ostlund et al (1982) and Feigenbaum et al (1982). We then state our results, followed by a description of the proof. We conclude with a brief discussion of some of the philosophical and mathematical implications of the method of proof. We relinquish most of the details of the proof to the appendices at the end.

More specifically, in Chapter 1 we give the physical motivation for the theory developed in Ostlund et al (1983). We review this theory in Chapter 2. Chapter 3 contains a statement of the results and a description of the proof. In Chapter 4 we give the basic ideas behind the numerical functional analysis used in the proof, while most of the details are left to the appendices. Chapter 5 contains a discussion of some of the philosophical and mathematical questions arising from this type of proof. The references are listed in Chapter 6.

Appendix 0 contains a brief list of some of the notation used elsewhere in the text. Appendix 1 contains a brief review of the theory of continued fractions, while Appendix 2 does the same for

circle maps. Appendix 3 contains a number of results on the spaces of analytic functions that we use. We discuss the notion of analyticity improving linear operators and show that they are compact operators. We list some results on perturbation theory for linear operators and discuss various aspects of the L^1-spaces that we use. Appendix 4 details various results on the spectrum of the derivative of the renormalisation transformation, and in Appendix 5 we list some basic results on stable and unstable manifolds. Appendix 6 contains various formulae connected with the renormalisation transformation and its derivative. Appendix 7 contains the computer program and some detailed explanatory notes. Finally, in Appendix 8 we discuss the error bounding routines and the implementation of numerical functional functional analysis on the computer.

1. Physical Motivation

The aim of this chapter to provide some physical motivation for the problem we shall discuss in Chapter 2. We shall assume a familiarity with the basic ideas of dynamical systems. These include the notions of flows, maps, orbits, hyperbolic sets, stable and unstable manifolds and attractors (see Palis and De Melo (1982) and Irwin (1980) for more information). In this account we shall not be concerned with mathematical rigour.

We are interested in parametrised dissipative dynamical systems on some (possibly infinite dimensional) manifold. For example, this manifold might be the space of velocity fields of a fluid with the flow corresponding to evolution under the Navier-Stokes equations. The parameter will be in general be some form of "stress" on the system or a measure of the non-linear departure from a simple system.
For each parameter value, there will be attractors that will govern the long term behaviour of the system. We are interested in the changes or bifurcations that take place in the attracting sets as we vary the parameter. Of particular interest is the transition from regular motion to turbulent or chaotic motion. Of course, there are many possibilities that can occur and we shall focus on one particular scenario, which, as we shall see, has been found in some physical systems.

One common bifurcation is the Hopf birfurcation (see Marsden and McCracken (1976)) in which an attractive stationary point of a flow destabilises and throws off an attractive periodic orbit. This is illustrated in Figure (1). Under further change in the parameter, this

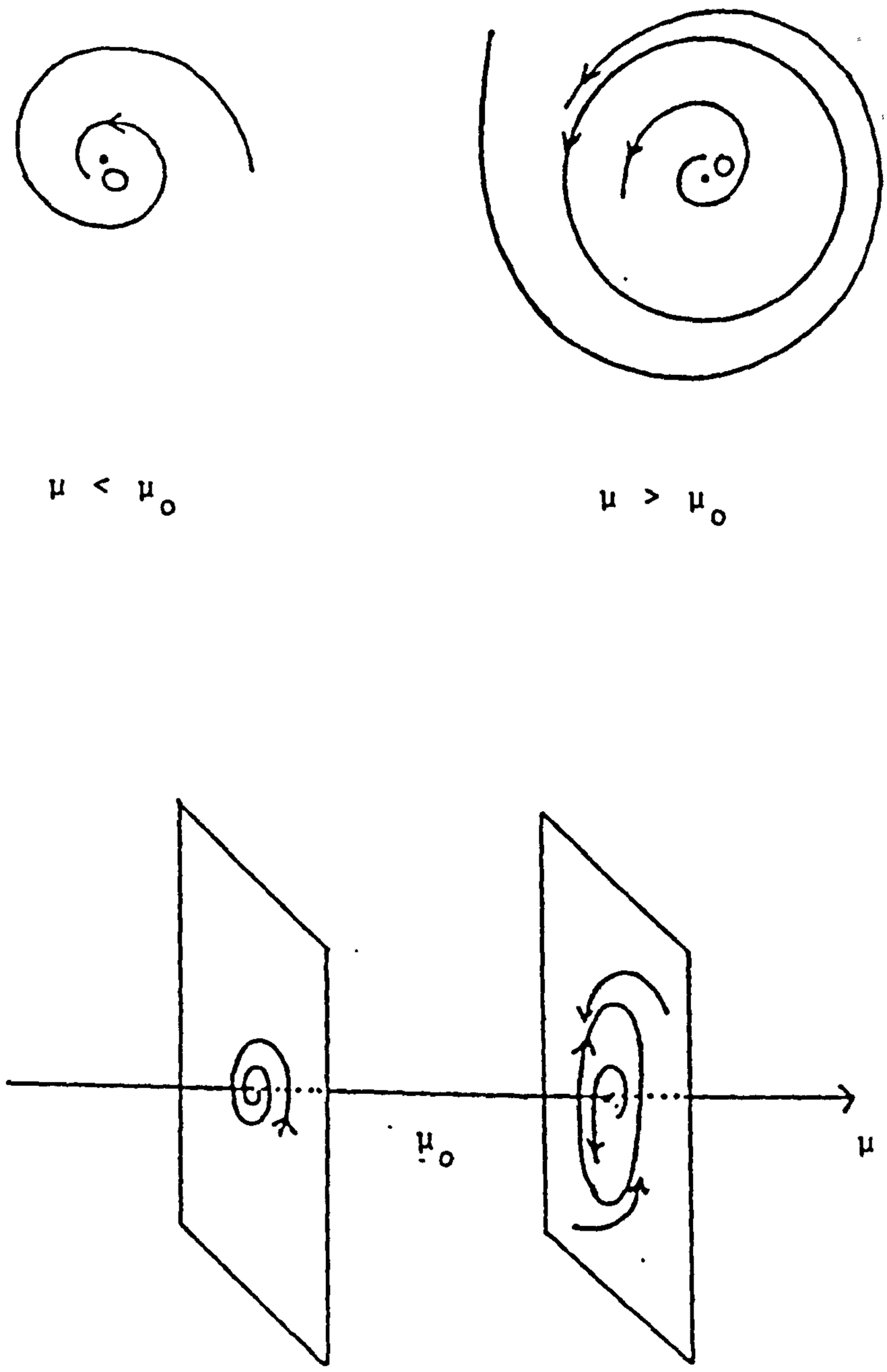


Figure 1. The Hopf bifurcation.
As the parameter μ is increased, an attractive stationary point for the flow destabilises and throws off a periodic orbit.
(Pictures from Zeeman (1982) and Whitley (1983).)

periodic orbit can itself become unstable and bifurcate to an attracting two dimensional torus (or 2-torus) (Figure (2)). The flow on the 2-torus is determined by two frequencies f_1, f_2. If their ratio f_1/f_2 is rational then the flow on the torus is "phase-locked." In this case there is an attractive periodic orbit within the torus. On the other hand, if the ratio is irrational then the flow on the torus is "quasi-periodic" (with two frequencies) and the orbits wrap densely round the torus.

In the 1950's Landau developed a theory for the onset of turbulence in fluids (Landau and Lifschitz (1959)). He supposed an infinite sequence of Hopf bifurcations producing quasi-periodic motion of ever increasing complexity. However Ruelle and Takens (1971) showed that the Landau model was not likely (at least in a topological sense) and suggested modelling chaotic motion by low dimensional "strange attractors" (i.e. attractors that were not stationary or periodic). The flow on these attractors shows "sensitive dependence on initial conditions," which means that orbits of points that are initially close together in phase space diverge quickly and after a while there is no correlation between them. This gives the effect of "randomness" in systems that are completely deterministic. Ruelle and Takens showed that a direct transition from quasi-periodicity on a 2-torus to chaotic motion was not an unlikely event. It is this transition that we shall be dealing with here. The basic model can be written as:

(stationary state ---->) periodic ---> quasi-periodic ---> chaotic

(1 frequency) (2 frequencies)

We shall now describe two physical sytems that display this type of

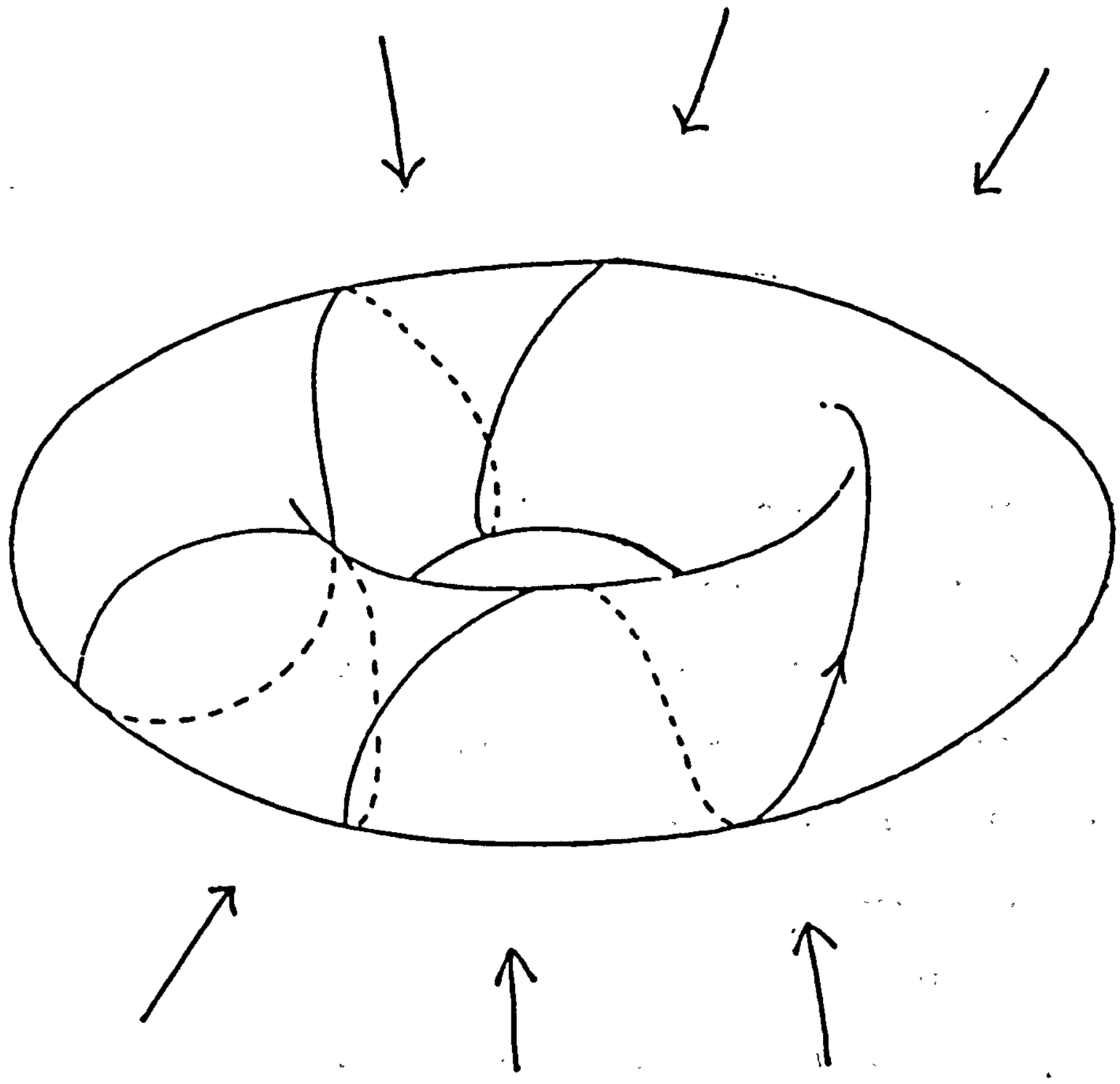


Figure 2. An attracting 2-torus in phase space. Orbits of nearby points are asymptotic to the torus. The flow on the torus is determined by two frequencies f_1 and f_2. If f_1/f_2 is rational, then the flow is phase locked. Otherwise the flow is quasi-periodic and orbits wrap densely round the torus.
(Picture from Jensen et al (1984).)

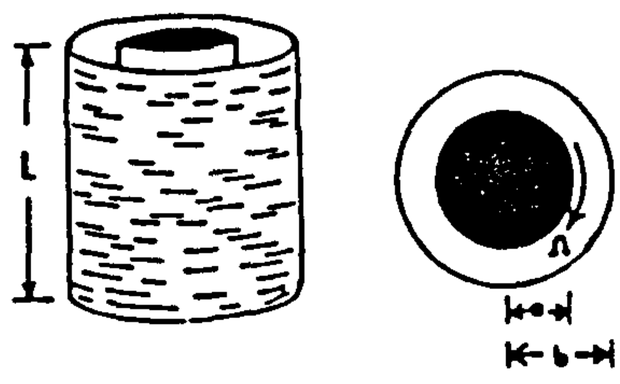
behaviour. They are Taylor vortex flow and Rayleigh-Bénard convection. They are illustrated in Figure (3). We shall only give a brief description here. A detailed account is given in Swinney and Gollub (1981).

(i) Taylor vortex flow

In this experiment, fluid is placed between two concentric cylinders and the inner cylinder is rotated with angular velocity Ω. This Ω is the "stress parameter" on the system. For small values of Ω the fluid rotates laminally around the inner cylinder. But for larger Ω, instabilities produce Taylor cells that can be seen clearly in Figure (5), which shows the motion of the fluid for various values of Ω. Inside the cells the fluid swirls as it rotates around the inner cylinder. The orientation of this swirling (vortex) motion alternates from cell to cell. However, although the motion is rather elaborate, at first the picture is of steady state flow. The flow pattern is stationary in time. As Ω is increased, the system bifurcates and the flow pattern modulates producing wavy vortex flow. At first, this modulation is periodic, but as Ω is further increased the modulation becomes quasi-periodic; the flow never quite returns to its original state. Further increase in Ω, produces turbulent flow, clearly visible in the last picture of Figure (5). However, although turbulent, there is still a great deal of order in the flow.

(ii) Rayleigh-Bénard convection

The Rayleigh-Bénard experiment consists of a rectangular cell containing fluid (Figure (3)). A temperature difference between the top and bottom faces of the cell produces convection currents in the fluid. The size of the temperature difference ΔT is the "stress"



(a) Taylor vortex flow

(b) Rayleigh-Benard convection flow

Figure 3. The experimental apparatus for Taylor vortex flow and for Rayleigh-Benard convection flow (Pictures from Swinney and Gollub (1978).)

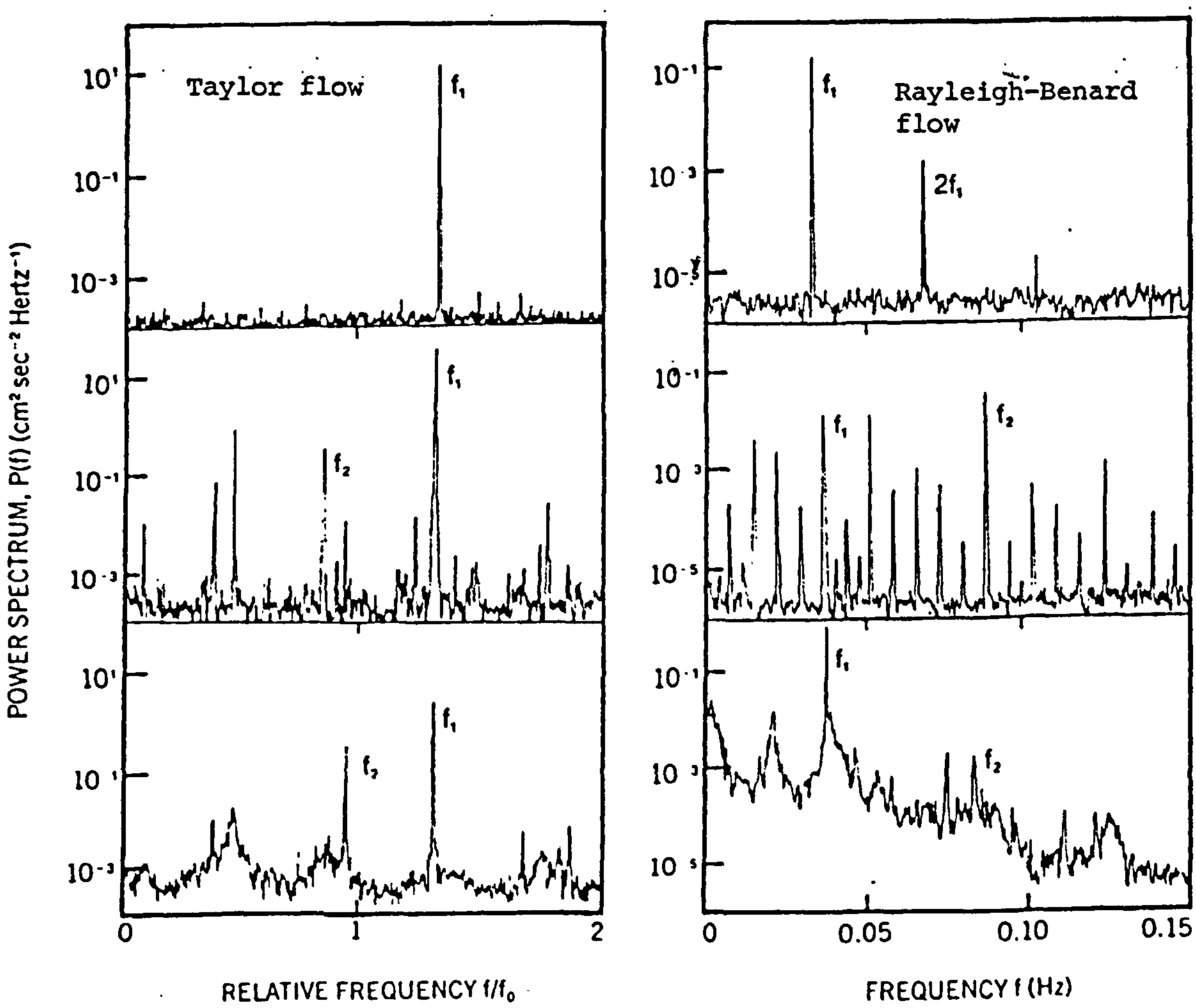
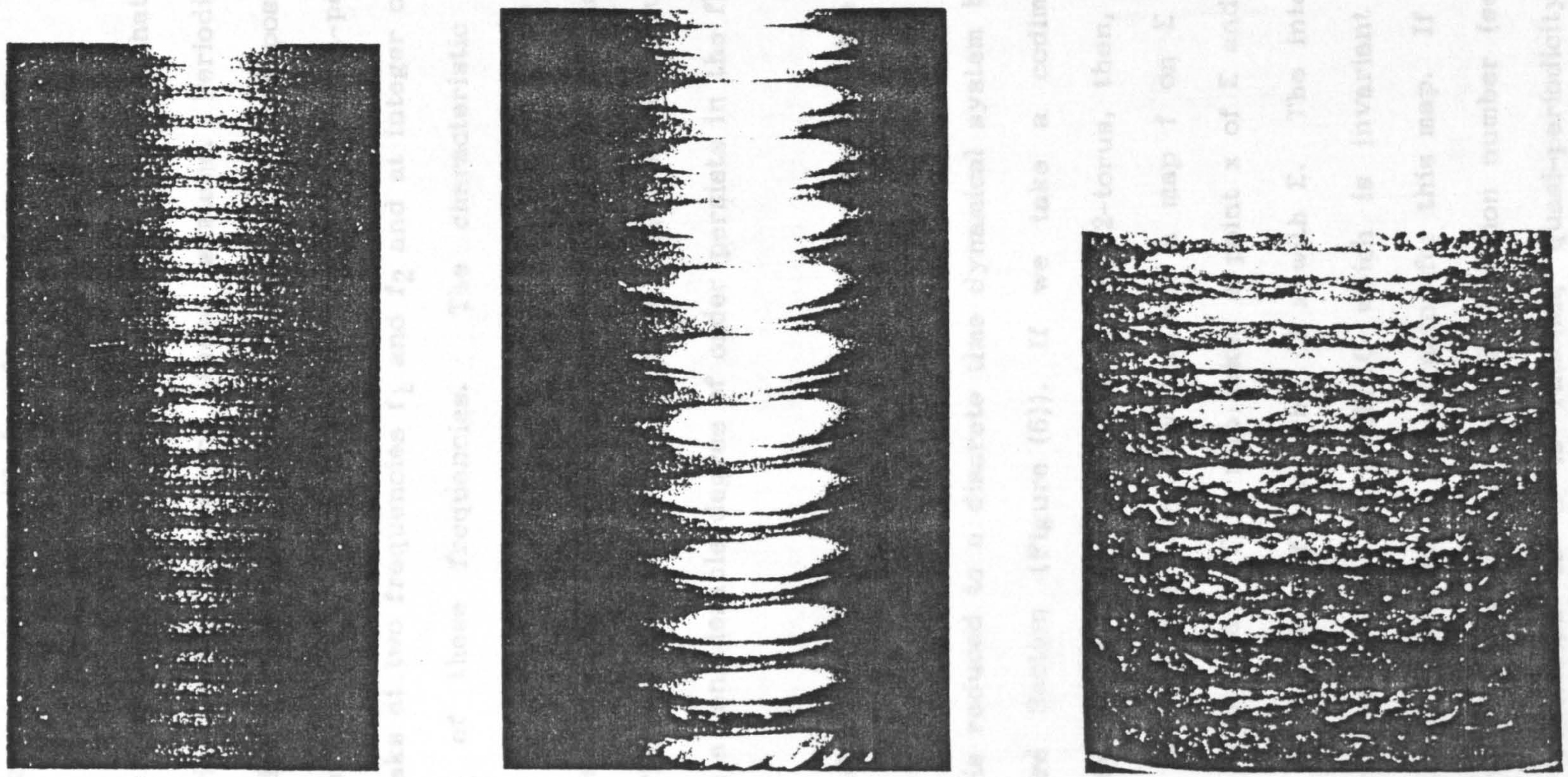


Figure 4. Spectra for Taylor vortex flow and Rayliegh-Benard convection flow. The top diagrams show periodic flow with one (fundamental) frequency f_1. The middle diagrams show quasi-period flow with two frequencies f_1, f_2, f_1/f_2 irrational. The bottom diagrams show weak turbulence with characteristic broad band spectrum. (Pictures from Swinney and Gollub (1978).)



(a) stationary flow (b) periodic flow (c) turbulent flow

Figure 5. Photographs of Taylor vortex flow. The Taylor cells may be seen in all three pictures. (Pictures taken from Di Prima and Swinney (1981).)

parameter. This system also bifurcates from periodic to chaotic motion through quasi-periodicty with two frequencies.

Referring to Figure (4), we see the spectral diagrams that come from time series measurements on these systems. Periodic flow is characterised by peaks at a single frequeny f_1 (and possibly at its harmonics $2f_1$, $3f_1$, ...). The spectral diagram for quasi-periodic flow shows peaks at two frequencies f_1 and f_2 and at integer combinations $mf_1 + nf_2$ of these frequencies. The characteristic feature of turbulent systems is broad band spectrum; a wide range of frequencies can be discerned above the background noise of the system. This type of behaviour is often called weak turbulence since sometimes a considerable degree of order persists in the flow.

In Ostlund et al (1983) a mechanism is suggested for the breakdown of the 2-torus in phase space producing turbulent flow. First of all the flow is reduced to a discrete time dynamical system by means of a Poincaré Section (Figure (6)). If we take a codimension one hypersurface Σ that is transverse to the 2-torus, then, sufficiently close to the 2-Torus, the flow induces a map f on Σ (called the Poincaré return map). This map takes a point x of Σ and maps it to the next intersection of the orbit of x with Σ. The intersection of the 2-torus with Σ is a circle C which is invariant under the Poincaré return map and is an attractor for this map. If the flow on the 2-torus is phase locked then the rotation number (see Appendix 2) of f restricted to C is rational; quasi-periodicity gives an irrational rotation number.

Ostlund et al (1983) propose the following mechanism for the

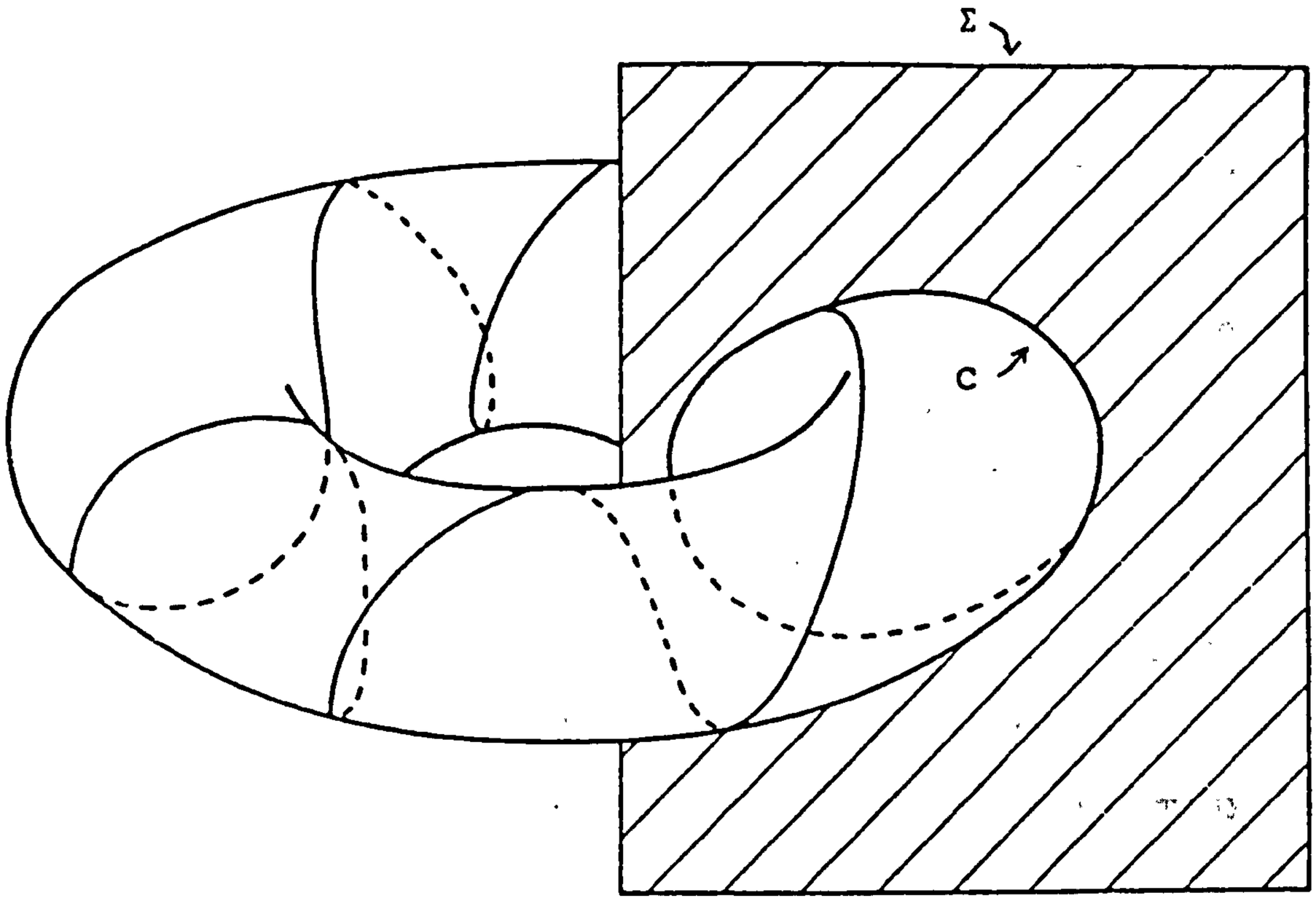
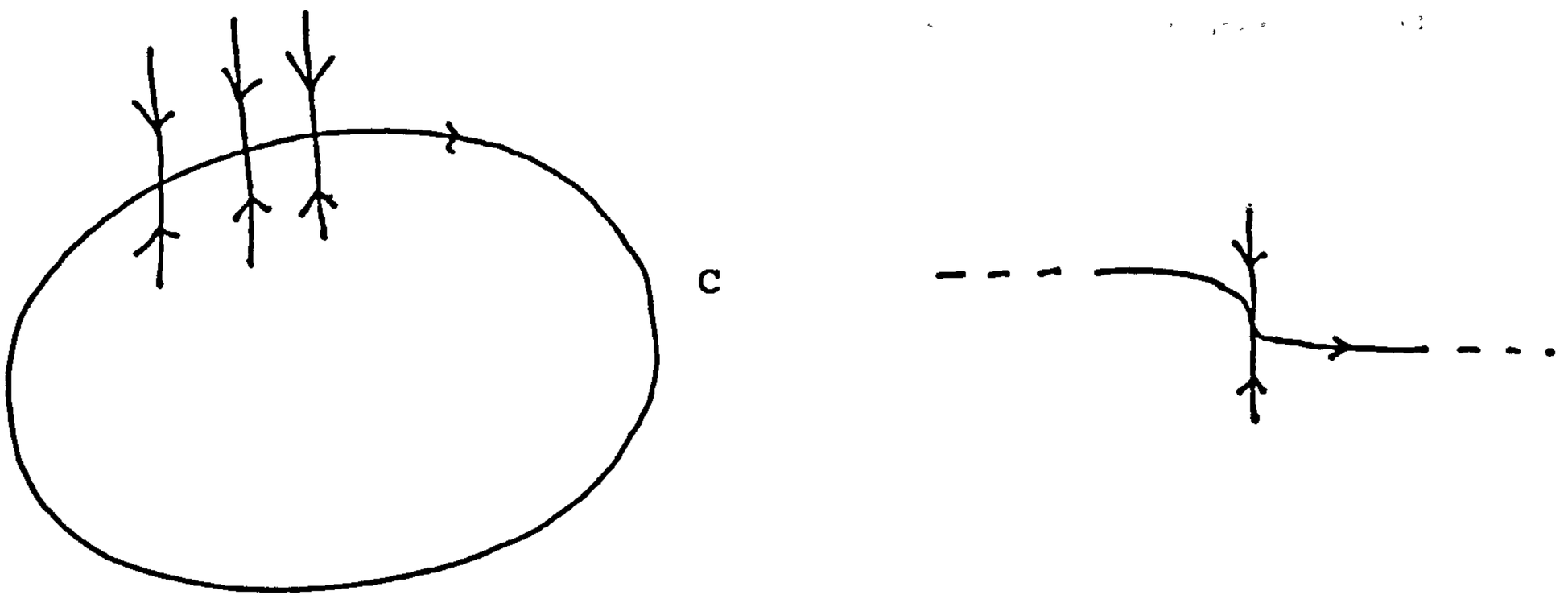


Figure 6. A Poincaré cross-section of an attracting 2-torus. The codimension one manifold Σ is transverse to the 2-torus. The intersection of Σ with the torus is a circle C which is attracting for the Poincaré return map. (Picture from Jensen et al (1984).)



(a) The strong stable foliation

(b) cubic tangency of C with the foliation.

Figure 7. The attracting invariant circle C for the Poincaré map and its strong stable foliation (a). (b) shows schematically a cubic tangency of C with the foliation. In fact such tangencies will be dense in C.

breakdown of this invariant circle. For any point x on the circle, we consider the set of points of Σ that converge together with x under application of f. These sets form codimension one manifolds in Σ and form the leaves of the strong stable foliation (SSF). Ostlund et al (1983) suggest that breakdown of the invariant circle occurs when the circle forms an inflection point with a leaf of this strong stable foliation. This is illustrated in Figure (7). This tangency will generically be cubic in nature and will induce a cubic critical point in the map on the circle. In fact, the action of the map will transmit the tangency around the circle and there will be infinitely many tangencies. Nevertheless, they propose to model this situation with parameterised families of maps of the circle that develop a single cubic singularity. They recognise that this is not realistic, but, as we shall see, Rand (1984) has been able to take the theory for maps of the circle, and extend it to higher dimensional systems.

In the next chapter, we discuss the renormalisation theory in Ostlund et al (1983).

2. Renormalisation of maps of the circle

Having provided some physical motivation for study of parametrised families of maps of the circle, we shall outline the theory developed in Ostlund et al (1983) and Feigenbaum et al (1982). This subject is of mathematical interest in its own right. We assume that the reader has a basic knowledge of circle maps viz. rotation number, Denjoy's theorem and the Herman-Yoccoz theorem. This basic theory is outlined in Appendix 2. A comprehensive account can be found in Herman (1976), Nitecki (1971) and Cornfeld et al (1982).

2.1 Parametrised families of maps of the circle

The dynamics of maps of the circle depends crucially on the rotation number ρ of the map. A map for which $\rho \in \mathbb{Q}$ has (generically) attractive periodic orbits, while C^2 diffeomorphisms with $\rho \in \mathbb{R} \setminus \mathbb{Q}$ are topologically conjugate to an ergodic rotation. Given a one parameter family of maps, then (generically) the rotation number will vary with the parameter. In order to keep the rotation number constant we shall need to introduce an extra parameter to "tune" the system. We therefore consider two parameter families of maps. One particular two parameter family of maps of the circle is the "sine" family, with two parameters ω, a, which is given (when lifted to $\mathbb{R}$) by

$$f_{\omega,a}(x) = x + \omega - a/2\pi \,.\, \sin(2\pi x) \qquad (2.1)$$

Here $a \in [0,1]$ and $\omega \in [0,1)$. This family is often used as a model two parameter family and displays generic behaviour (apart from its obvious symmetry about the point $x = \frac{1}{2}$). If $a = 0$, then $f_{\omega,0}$ is a rotation through a constant angle ω. For $a \in [0, 1)$, $f_{\omega,a}$ is a diffeomorphism, while $f_{\omega,1}$ has a single cubic critical point at 0. Thus $a \to 1$ in this model familiy represents the breakdown of the

invariant 2-torus according to the scheme in Ostlund et al (1982). The parameter "a" is a non-linearity parameter, while the parameter "ω" is the frequency of a driving force. Together they determine the rotation number of the system.

For $a \in [0,1)$, the picture is as illustrated in Figure (8). This picture was described by Arnold (1965). The figure shows the regions of the (ω,a)-plane for which the rotation number $\rho(f_{\omega,a})$ is rational. These are the Arnold tongues. Taking a cross section for constant $a > 0$, the set of ω for which ρ is rational has non-zero measure and increases as a increases. It is conjectured that for $a = 1$ this set has full measure. Figure (9) shows the graph of $\rho(f_{\omega,a})$ against $2\pi\omega$ for $0 < a < 1$. The picture shows the "Devils staircase" function. ρ is constant at every rational value and increasing at every irrational value. For Ω irrational, the function $\omega(a)$, determined so that $\rho(f_{\omega(a),a}) = \Omega$, is a curve from the line $a = 0$ to $a = 1$.

A dissipative map of the plane that is related to $f_{\omega,a}$ is the dissipative sine map defined by:

$$\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} x + \omega + \lambda.y - a/2\pi.\sin 2\pi x \\ \lambda.y - a/2\pi.\sin 2\pi x \end{bmatrix}, \quad 0 < \lambda < 1 \qquad (2.2)$$

λ is the dissipation parameter. For $\lambda = 0$, the map reduces to the map $f_{\omega,a}$ when one restricts to the y-axis. For $\lambda = 1$, we obtain the area-preserving map called the standard or Taylor-Chirikov map. This has been extensively studied in Hamiltonian Mechanics. For $0 < \lambda < 1$, the map exihibits the universal behaviour in the breakdown of its invariant tori (see Bohr (1984)).

2.2 Scaling Laws for Circle Maps

Following Feigenbaum's work on universal scaling laws for period

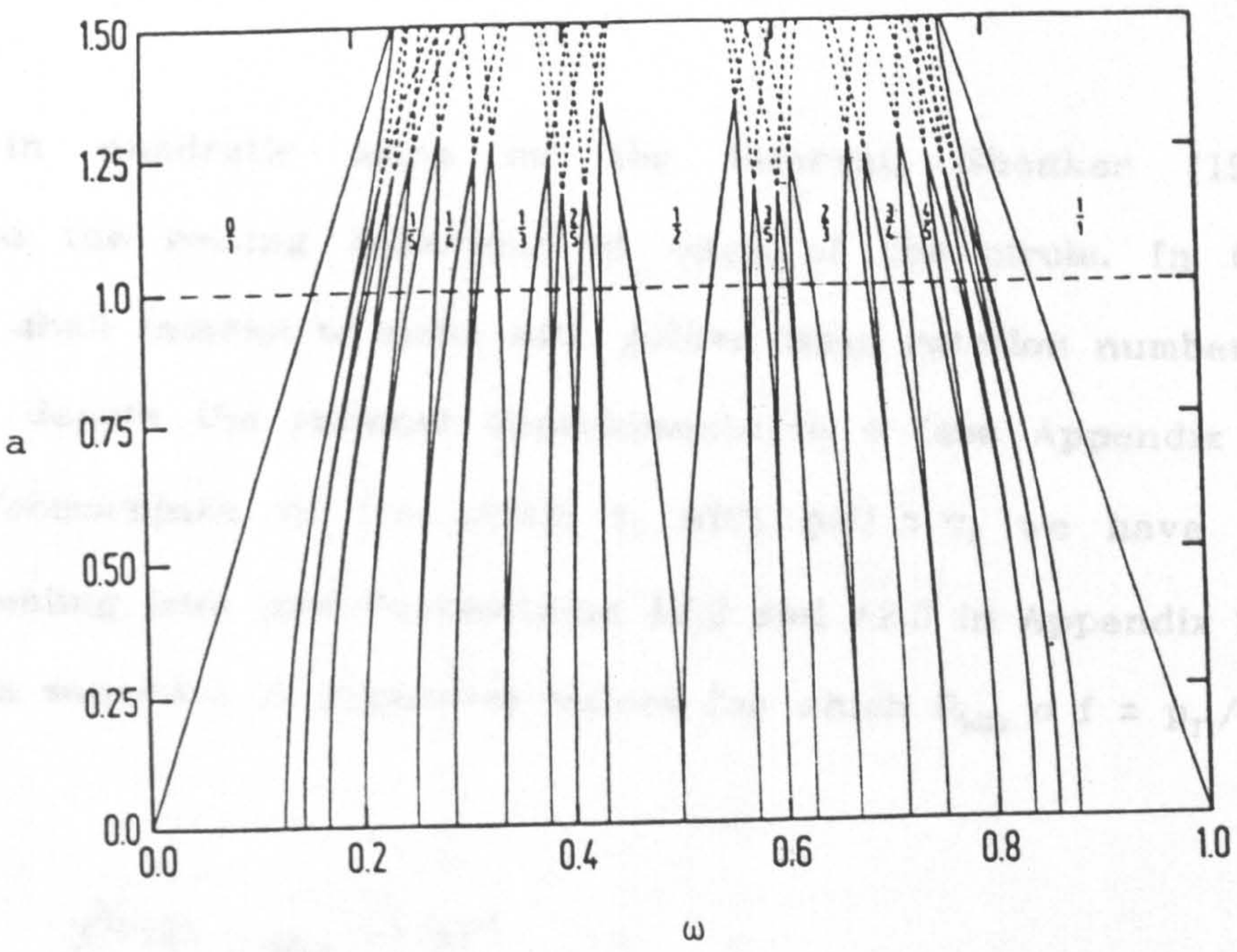


Figure 8. The Arnold tongues for the sine map $f_{\omega,a}$. The diagram shows the regions of the (ω,a)-plane for which $\rho(f_{\omega,a})$ is rational. (Picture from Jensen et al (1984).)

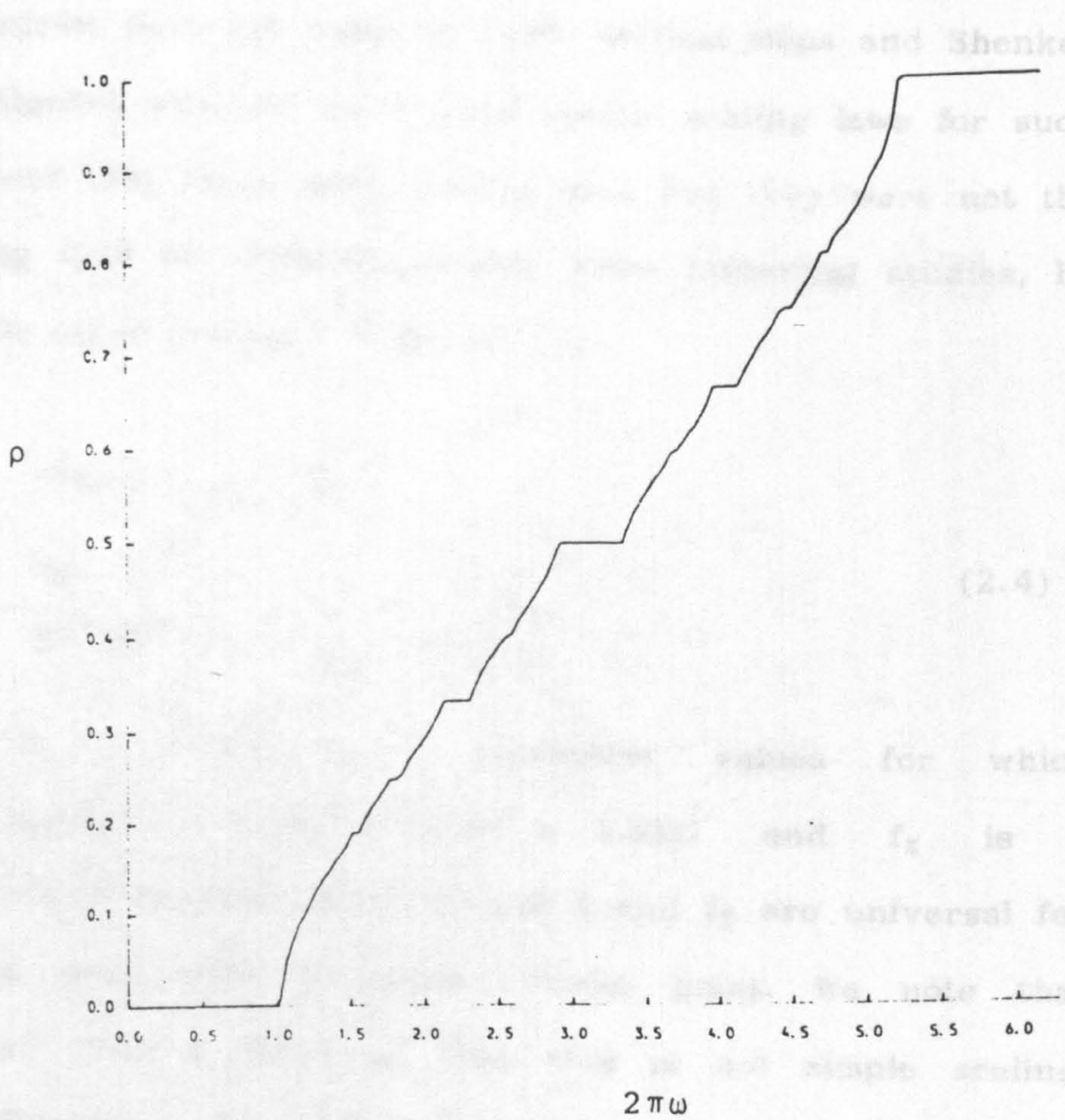


Figure 9. Graph of rotation number $\rho(f_{\omega,a})$ against $2\pi\omega$ for fixed $a > 0$. This "Devils staircase" is constant at every rational ρ and increasing at every irrational ρ.

doubling in quadratic maps on the interval, Shenker (1982) investigated the scaling behaviour of maps of the circle. In this section we shall restrict to maps with golden mean rotation number σ. Let p_n/q_n denote the rational approximants to σ (see Appendix 1). For a diffeomorphism of the circle f, with $\rho(f) = \sigma$, we have the following scaling laws (see Propositions A2.2 and A2.3 in Appendix 2). Let ω_n be a sequence of parameter values for which $R_{\omega n} \circ f = p_n/q_n$. Then:

$$\begin{aligned} &\text{(i)} && f^{q_n}(0) - p_n \sim (-\sigma)^n \\ &\text{(ii)} && \omega_n \sim (-1/\sigma^2)^{-n} \\ &\text{(iii)} && (-\sigma)^{-n}(f^{q_n}((-\sigma)^n x) - p_n \to R_\alpha \text{ some } \alpha \in \mathbb{R}. \end{aligned} \tag{2.3}$$

We shall refer to these laws as simple scaling.

Herman's Theorem does not apply to cubic critical maps and Shenker (1982) investigated whether there were similar scaling laws for such maps. He found that there were scaling laws but they were not the simple scaling laws for diffeomorphisms. From numerical studies, he found that for cubic critical f with $\rho(f) = \sigma$

$$\begin{aligned} &\text{(i)} && f^{q_n}(0) - p_n \sim \beta^n \\ &\text{(ii)} && \omega_n' \sim \delta^{-n} \\ &\text{(iii)} && \beta^{-n}(f^{q_n}(\beta^n x) - p_n) \to f_*(x) \end{aligned} \tag{2.4}$$

where ω_n' is a sequence of parameter values for which $\rho(R_{\omega n} \circ f) = p_n/q_n$. $\beta \cong -0.776051$, $\delta^{-1} \cong -2.8331$ and f_* is a non-linear analytic function of x^3. The β, δ and f_* are universal for cubic critical maps with a single critical point. We note that $\sigma \cong -0.618$ and $-1/\sigma^2 \cong -2.618$ so that this is not simple scaling. Figure (10) illustrates the scaling laws (2.3)(ii) and (2.4)(ii) for the sine map $f_{\omega,a}$. Here the ω_n and ω_n' are the distances of the Arnold

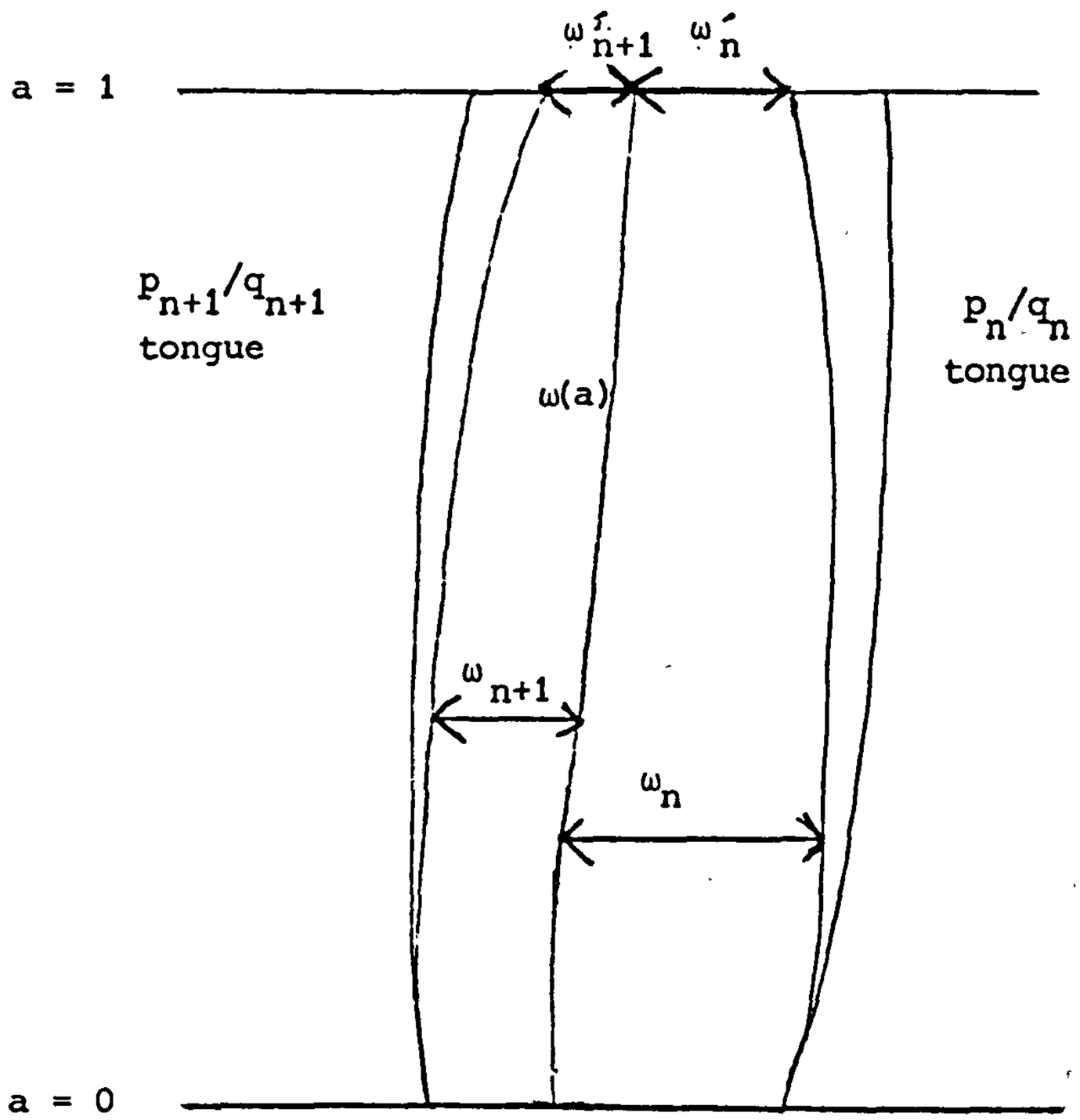


Figure 10. Scaling behaviour of Arnold tongues for the sine map $f_{\omega,a}$. ω_n, ω'_n are the distances of the p_n/q_n tongue from the curve $\omega(a)$, on which the rotation number is σ. For $0 < a < 1$, $\omega_n \sim (-1/\sigma^2)^{-n}$, while $\omega'_n \sim \delta^{-n}$, where $\delta \cong -2.813$.

tongues from the curve $\rho(f_{\omega(a),a}) = \sigma$. For $0 < a < 1$, $\omega_n \sim (-1/\sigma^2)^{-n}$, but for $a = 1$, when $f_{\omega,a}$ is cubic critical, $\omega_n' \sim \varsigma^{-n}$, where $\varsigma \cong -2.8331$.

We investigate the functional equation that f_* satisfies. For $\rho(f) = \sigma$, we have the relations $q_{n+1} = q_n + q_{n-1}$, $p_{n+1} = p_n + p_{n-1}$, so that

$$\beta^{-(n+1)}(f^{q_{n+1}}(\beta^{n+1}x) - p_{n+1}) =$$

$$\beta^{-1}\beta^{-n}(f^{q_n}(\beta^n.\beta^{-1}.\beta^{-(n-1)}(f^{q_{n-1}}(\beta^{n-1}\beta^2 x) - p_{n-1})) - p_n).$$

Taking the limit as $n \to \infty$ gives the functional equation for f_*:

$$\beta^{-1}f_*(\beta^{-1}f_*(\beta^2 x)) = f_*(x) \qquad (2.5)$$

Similarly, writing $q_{n+1} = q_{n-1} + q_n$ and $p_{n+1} = p_{n-1} + p_n$ gives the equation:

$$\beta^{-2}f_*(\beta.f_*(\beta x)) = f_*(x) \qquad (2.6)$$

(We note that in Ostlund et al (1983) and Feigenbaum et al (1982) these equations are written in terms of $\alpha = \beta^{-1}$.)

Nauenberg (1982) has shown that (under suitable assumptions) any analytic solution of one of these two equations is also a solution of the other.

We note that the function R_α satisfies both of these equations (2.5) and (2.6) with β set equal to $-\sigma$. The scaling hypothesis ((2.4)(iii)) is certainly enough to explain the first scaling law (2.4)(i) above. However to obtain (2.4)(ii) we need to enlarge the function space and consider pairs of functions. Although there is a similar method contained in Feigenbaum et al (1982) we shall describe the scheme contained in Ostlund et al (1983).

2.3 Pairs of maps

We continue to restrict to the golden mean rotation number. For

$\rho(f) = \sigma$, we have $p_{n+1} = q_n = F_n$, where F_n is the nth Fibonacci number ($F_0 = 0$, $F_1 = 1$, ...). The Fibonacci numbers satisfy the recurrence relations

$$F_{n+1} = F_n + F_{n-1} \tag{2.7}$$

and

$$F_{n+1} = F_{n-1} + F_n \tag{2.8}$$

These recurrence relations are second order. It is possible to write (2.7) as a first order recurrence relation on a two dimensional space as follows:

$$\begin{bmatrix} F_n \\ F_{n+1} \end{bmatrix} = T^n \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad T : \begin{bmatrix} A \\ B \end{bmatrix} \longrightarrow \begin{bmatrix} B \\ B + A \end{bmatrix} \tag{2.9}$$

We can also generate Fibonacci iterates by the same process

$$\begin{bmatrix} f^{F_n} \\ f^{F_{n+1}} \end{bmatrix} = T^n \begin{bmatrix} f \\ f \end{bmatrix}, \quad T : \begin{bmatrix} \xi \\ \eta \end{bmatrix} \longrightarrow \begin{bmatrix} \eta \\ \eta \circ \xi \end{bmatrix} \tag{2.10}$$

We can also write (2.8) in the same manner:

$$\begin{bmatrix} F_n \\ F_{n+1} \end{bmatrix} = T^n \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad T : \begin{bmatrix} A \\ B \end{bmatrix} \longrightarrow \begin{bmatrix} B \\ A + B \end{bmatrix} \tag{2.11}$$

giving the following for iterates:

$$\begin{bmatrix} f^{F_n} \\ f^{F_{n+1}} \end{bmatrix} = T^n \begin{bmatrix} f \\ f \end{bmatrix}, \quad T : \begin{bmatrix} \xi \\ \eta \end{bmatrix} \longrightarrow \begin{bmatrix} \eta \\ \xi \circ \eta \end{bmatrix} \tag{2.12}$$

However, while these two methods for generating the Fibonacci numbers are identical, the ones for functions are not, unless the pair of functions (ξ, η) commute i.e. $\xi \circ \eta = \eta \circ \xi$. This phenomenon will introduce extra eigenvalues of the renormalisation transformation. Following Ostlund et al (1983) we shall use the first of these recurrence schemes.

The essential idea of Ostlund et al (1983) is to work with pairs of maps that glue together to give a map of the circle. Denote by S a space of pairs (ξ, η) of maps on $\mathbb{R}$ with the following properties:-

$$\begin{aligned} &\text{(i)} \quad \xi(0) = \eta(0) + 1 \\ &\text{(ii)} \quad \eta(0) < 0 < \xi(0) \\ &\text{(iii)} \quad \xi(\eta(0)) = \eta(\xi(0)) \\ &\text{(iv)} \quad \xi, \eta \text{ are increasing on } [\eta(0),0] \text{ and } [0,\xi(0)] \\ &\qquad \text{respectively.} \end{aligned} \tag{2.15}$$

Now we define

$$f_{\xi,\eta} = \begin{cases} \xi & \text{on} \quad [\eta(0),0] \\ \eta & \text{on} \quad [0,\xi(0)] \end{cases} \tag{2.14}$$

Then $f_{\xi,\eta}$ induces a map on the circle obtained by identifying the points $\eta(0)$ and $\xi(0)$. We note also that maps of the circle can be embedded in this space of maps. For if f is a lift of a circle map, then writing

$$\begin{aligned} \xi_f &= f \\ \eta_f &= f - 1 \end{aligned} \tag{2.15}$$

we obtain a pair of maps that give f when glued together again. Note that since this pair comes from a single map f, the functions ξ_f, η_f must commute i.e $\xi_f \circ \eta_f = \eta_f \circ \xi_f$. The construction of a circle map from a pair of maps is illustrated in Figure (11).

It is possible to define a rotation number for a pair $(\xi, \eta) \in S$ by the formula

$$\rho(\xi, \eta) = \rho(f_{\xi,\eta}). \tag{2.16}$$

2.4 Renormalisation Transformation

We aim to apply equation (2.10) to obtain a transformation of pairs of functions (ξ, η) that has a fixed point corresponding to solutions of

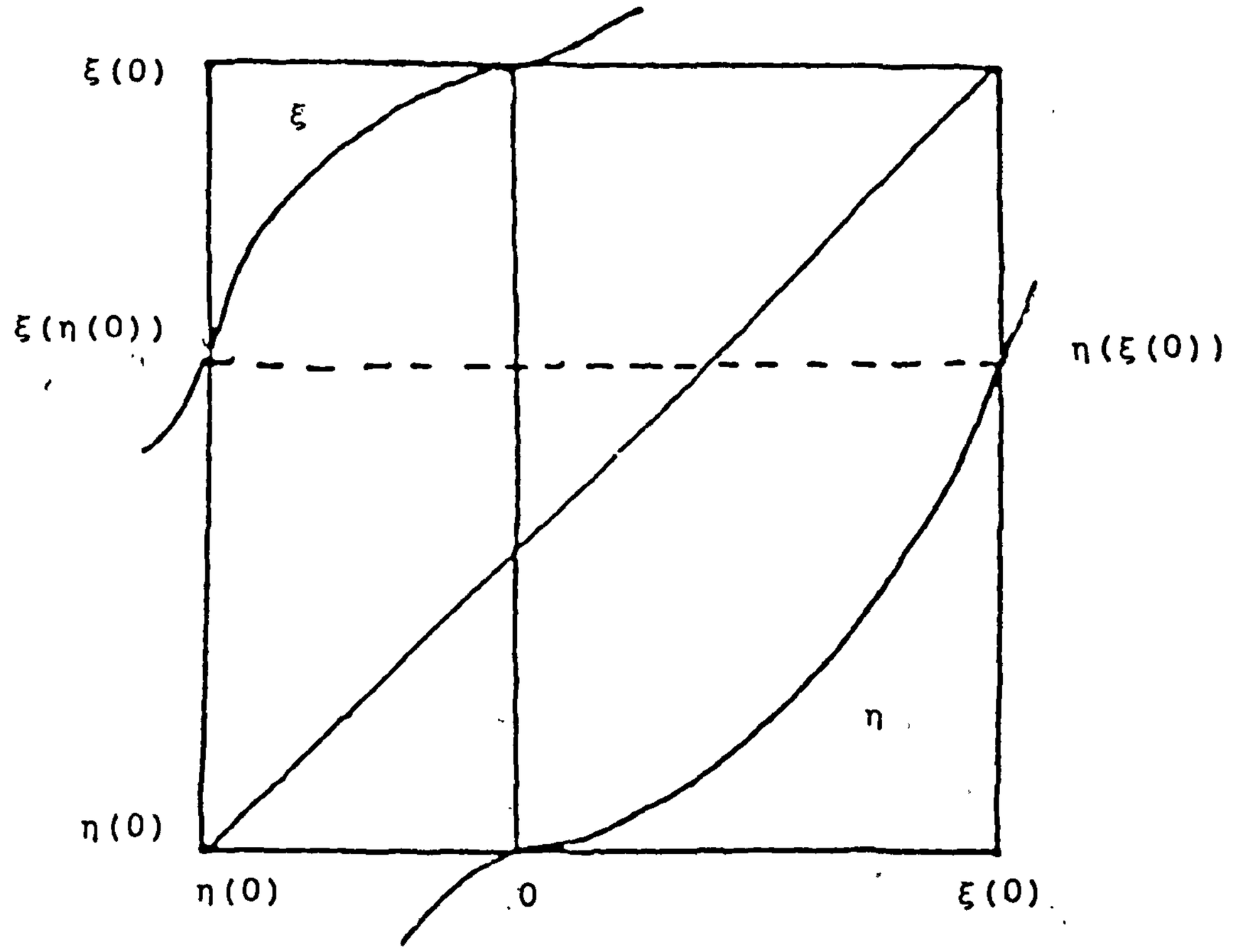


Figure 11. Constructing a circle map from of pair of functions (ξ,η). If $\xi(\eta(0)) = \eta(\xi(0))$, then the map $f_{\xi,\eta}$ defined as ξ on $[\eta(0),0]$ and η on $[0,\xi(0)]$ defines a map on the circle obtained by identifying the points $\eta(0)$ and $\xi(0)$.

equations (2.5) and (2.6). We need the following lemma.

Lemma 2.1 Let $(\xi, \eta) \in S$ with $\rho(\xi, \eta)$ irrational and with continued fraction expansion $[n, \ldots]$, $n > 0$. Let $I_0 = [0,\xi(0)]$. We write $f = f_{\xi,\eta}$. Then

$$f^i(I_0) \cap I_0 = \emptyset \text{ for } 0 \leq i < n,\ f^n(I_0) \cap I_0 \neq \emptyset$$

and

$$\xi^i(\eta(0)) < 0 \quad \text{for } 0 \leq i < n,\ \xi^n(\eta(0)) > 0.$$

proof We recall (Appendix 2) that $\rho = \rho(\xi, \eta)$ can be defined as the average number of points of the orbit of a point x that lies in $I_0 = [0,\xi(0)]$. Note that

$$1/(n+1) < \rho < 1/n$$

and that for $i > 0$, $f^i(I_0) \cap I_0 = \emptyset$ iff $\rho < 1/(i+1)$. The first conclusion follows simply from this. The second conclusion follows from the first by noting that the ith image of I_0 under f is $[\xi^{i-1}(\eta(0)), \xi^i(\eta(0))]$.

□

Definition Let $(\xi, \eta) \in S$ with $\rho(\xi, \eta) = [n, \ldots]$. Then we define the (renormalisation) transformation T_n by:

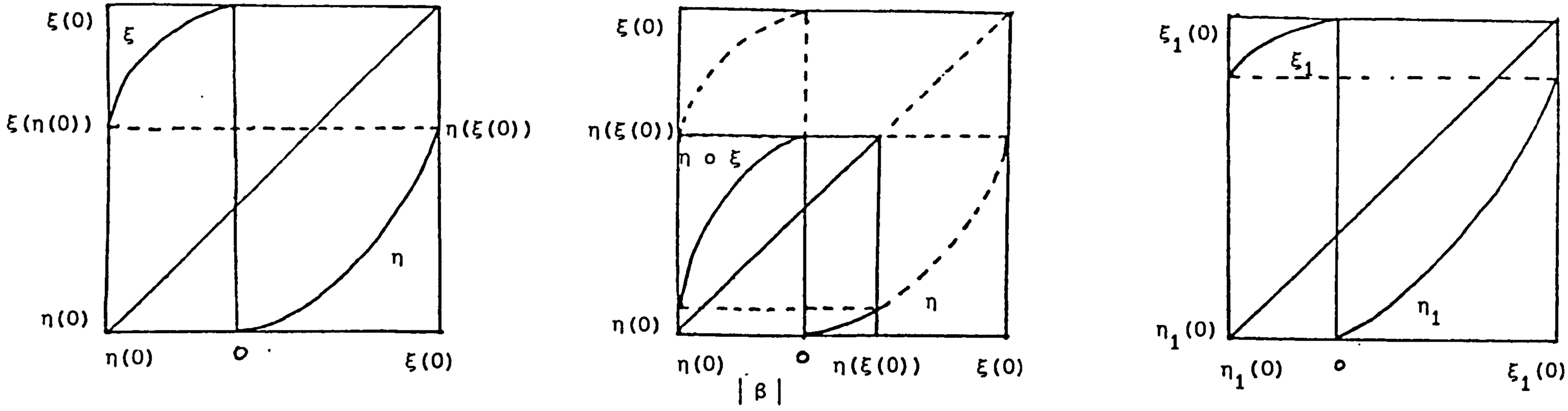
$$T_n : \begin{bmatrix} \xi \\ \eta \end{bmatrix}(x) \longrightarrow \begin{bmatrix} \beta^{-1}.\xi^{n-1}(\eta(\beta.x)) \\ \beta^{-1}.\xi^{n-1}(\eta(\xi(\beta.x))) \end{bmatrix} \qquad (2.17)$$

where $\beta = \xi^{n-1}(\eta(0)) - \xi^{n-1}(\eta(\xi(0)))$.

We note that in view of Lemma 2.1, T_n is well defined. For $n = 1$, we obtain:

$$T_1 : \begin{bmatrix} \xi \\ \eta \end{bmatrix}(x) \longrightarrow \begin{bmatrix} \beta^{-1}.\eta(\beta.x) \\ \beta^{-1}.\eta(\xi(\beta.x)) \end{bmatrix} \qquad (2.18)$$

where $\beta = \eta(0) - \eta(\xi\eta(0))$. This transformation T_1 is illustrated in Figure (12). The pair $(\eta \circ \xi, \eta)$ form a map of smaller circle of length $|\beta| = \eta(\xi(0)) - \eta(0)$. Scaling by $\beta = -|\beta|$ renormalises the circle



$$\xi_1(x) = \beta^{-1}\eta(\beta x)$$
$$\eta_1(x) = \beta^{-1}\eta(\xi(\beta x))$$

Figure 12. The renormalisation transformation T for golden mean rotation number. (a) The pair (ξ,η) define a map on a circle of length 1. (b) The pair $(\eta \circ \xi,\eta)$ define a map of a circle of length $|\beta|$. (c) Scaling by β restores the length to one and the asymmetry of the pair (ξ,η). (Picture from Rand (1983).)

to unit length. This is why the transformation T_1 (and also T_n) is called a "renormalisation" transformation. The scaling factor β is chosen negative in order to restore the asymmetry of (ε, η) for pairs of maps close to golden mean rotation number. For $f = (\varepsilon_f, \eta_f)$ we write $T_n(f)$ for $T_n(\varepsilon_f, \eta_f)$. The important property of the transformation T_n is the following.

Lemma 2.2 (Ostlund et al (1983)) Let $(\varepsilon, \eta) \in S$ and $\rho(\varepsilon, \eta) = [n, \ldots]$. Then

$$\rho(T_n(\varepsilon, \eta)) = 1/\rho(\varepsilon, \eta) - n \qquad (2.19)$$

proof Let $f = f_{\varepsilon,\eta}$. Let $I_0 = [0, \varepsilon(0)]$, $I_\ell = [\varepsilon^{\ell-1}(\eta(0)), \varepsilon^{\ell}(\eta(0))]$, for $\ell = 1, \ldots, n$, $J_1 = [\varepsilon^{n-1}(\eta(0)), 0]$ and $J_2 = [0, \varepsilon^n(\eta(0))]$. Choose x in J_2 such that $f^k(x) \neq 0$ for all k and let $m(k)$ denote the number of elements $x, f(x), \ldots, f^{k-1}(x)$ in I_0 and hence the same number in I_ℓ, $\ell = 1, \ldots, n$ (Lemma 2.1). Then $\rho(f) = \lim_{k\to\infty} m(k)/k$ (Appendix 2). Now chose a sequence k_i such that $k_i < k_{i+1}$ and $f^{k_i}(x) \in J_1$. Let $y = \beta^{-1}.x$, $\beta = \varepsilon^{n-1}(\eta(0)) - \varepsilon^{n-1}(\eta(\varepsilon(0)))$. Then there are precisely $m(k_i)$ elements of the sequence $x, \ldots, f^{k_i-1}(x)$ in I_0 and hence the same number in I_ℓ, $\ell = 1, \ldots, n$. Thus there are $k_i = n.m(k_i)$ in J_1. Consequently, $k_i - n.m(k_i)$ points of the sequence $y, T_n(f)(y), \ldots, T_n(f)^{m(k_i)-1}(y)$ fall in $I_0(T_n(f))$ which proves that

$$\rho(T_n(f)) = \lim (k_i - n.m(k_i))/m(k_i) = 1/\rho(f) - n.$$

□

Note that for $\rho(\varepsilon, \eta) = [n, n, n, \ldots]$ the transformation T_n leaves ρ unchanged. In particular, for $\rho(\varepsilon, \eta) = \sigma = [1, 1, 1, \ldots]$, $\rho(T_1(\varepsilon, \eta)) = \rho(\varepsilon, \eta) = \sigma$ and if $\rho(\varepsilon, \eta) = F_n/F_{n+1}$ for some n, then

$$\rho(T_1(\varepsilon, \eta)) = F_{n-1}/F_n. \qquad (2.20)$$

For $(\varepsilon, \eta) \in S$, we write T for the renormalisation transformation $T_1(\varepsilon, \eta)$ where $\rho(\varepsilon, \eta) = [1, \ldots]$.

Following Feigenbaum (1978) we look for fixed and periodic points for

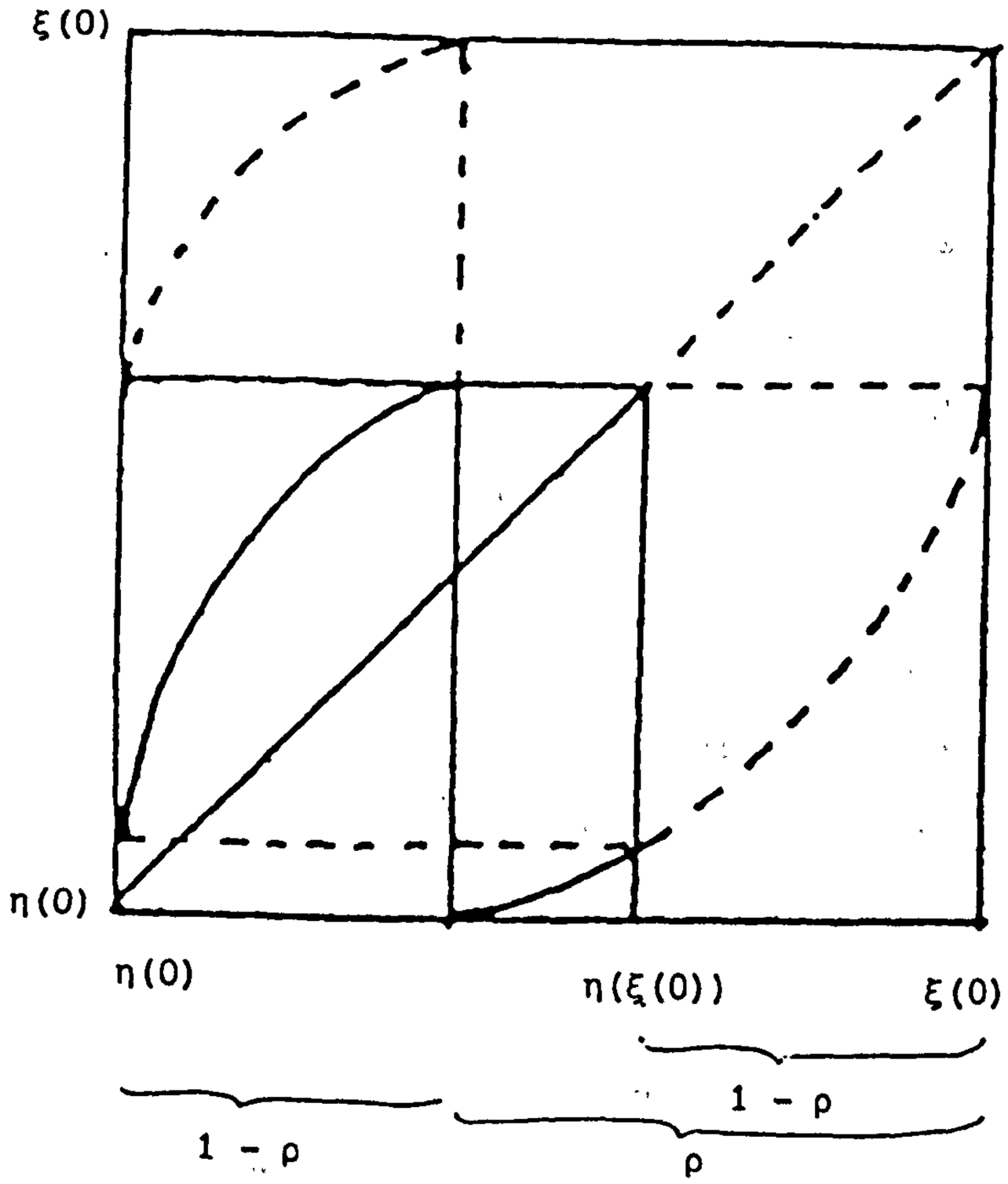


Figure 13. The effect of T_1 on the rotation number of a pair (ξ,η). Let x be such that $f^n_{\xi,\eta}(x) \neq 0$ for any n. Then the ratio of points in $[0,\xi(0)]$ is ρ. For the renormalised pair, this becomes the ratio of the number of points of the orbit in $[\eta(0),0]$ to the number in $[\eta(0),\eta(\xi(0))]$ which is $(1 - \rho)/\rho = 1/\rho - 1$.

T. Note that in view of Lemma 2.2 a fixed or periodic point of T must have a rotation number with a periodic continued fraction expansion. These rotation numbers are all quadratic irrationals (see e.g. Khinchin (1964)). We restrict ourselves to the simplest case which is when the continued fraction expansion has all entries equal to 1. This is the case of golden mean rotation number. Apart from its simplicity, we consider this case because σ is the number least well approximable by rationals for a given size of denominator (Khinchin (1964)). This means that in real physical systems this frequency is often the easiest to isolate from phased locked systems. The 2-torus with quasi-periodic motion with golden mean frequency ratio is often the last to break down as the stress parameter is increased. Also of importance are rotation numbers with continued fraction "tails" ending in 1's i.e. numbers of the form $[n_1, n_2, ,..., n_k, 1, 1, 1, ...]$ McKay (1982) has christened these "noble" numbers. The renormalisation theory for golden mean rotation number will also apply to these noble numbers, as applying T a number of times will strip the "head" of the number leaving the golden "tail."

From now on we shall implicitly use T to mean T_1, as defined in equation (2.18). We shall consider fixed points of T.

2.5 Simple or Weak Coupling Fixed Point.

There is a simple linear fixed point (ξ_s, ξ_s) of T given by:

$$(\xi_s, \eta_s) = (x + \sigma, x - \sigma^2). \qquad (2.21)$$

It is a simple matter to check that (ξ_s, η_s) is indeed a fixed point of T. This fixed point is studied extensively in Jonker and Rand (1983). They also develop a perturbation theory in the order of the singularity at 0 and have obtained fixed points that are analytic functions of $x.|x|^{\epsilon}$ for $\epsilon > 0$ sufficiently small. We list some of the

properties of this simple fixed point in Table 2.1.

Table 2.1 Properties of the simple fixed point

(a) $(\xi_s, \eta_s) = (x + \sigma, x - \sigma^2)$

(b) $\beta = -\sigma$

(c) T is well defined and C^∞ on a neighbourhood of (ξ_s, η_s) in a suitable space of pairs of functions (ξ, η) that satisfy:

(i) $\xi(\eta(0)) - \eta(\xi(0)) = 0$

(ii) $D(\xi \circ \eta - \eta \circ \xi)(0) = 0$

(d) $dT_s = dT(\xi_s, \eta_s)$ is a compact operator and the spectrum of dT_s consists of a simple eigenvalue $\delta = -1/\sigma^2$ and all of the rest of the spectrum of dT_s is contained within the disc $\{z : |z| < \sigma\}$

(e) There is a codimension one stable manifold and a one dimensional unstable manifold (see Appendix 5). The unstable manifold is the line $\mu \to (\xi_s + \mu, \eta_s + \mu)$.

From the discussion below, we shall see how the scaling behaviour (2.3)(ii) of diffeomorphisms with golden mean rotation number may be deduced from the hyperbolic structure of T at this simple fixed point.

2.6 Critical or Strong Coupling Fixed Point

In Chapter 3 we shall prove the existence of another fixed point (ξ_*, η_*) with ξ_*, η_* analytic functions of x^3. They are defined on domains Ω_1, Ω_2 of $\mathbb{C}$ specified in Chapter 3. The main properties of this critical fixed points are summarised in Table 2.2.

Table 2.2 Properties of the Critical Fixed Point

(a) ξ_*, η_* are non-trivial real analytic functions of x^3, defined on two domains Ω_1, $\Omega_2 \subseteq \mathbb{C}$.

(b) $\beta_* \cong -0.77605138$

(c) T is well defined and C^∞ on a neighbourhood of (ξ_*, η_*) in a suitable function space of pairs of maps (ξ, η) defined on Ω_1, Ω_2 respectively and which satisfy:

(i) $\xi(\eta(0)) - \eta(\xi(0)) = 0$

(ii) $D(\xi \circ \eta - \eta \circ \xi)(0) = 0$

(iii) $D^2(\xi \circ \eta - \eta \circ \xi)(0) = 0$

(iv) $D^3(\xi \circ \eta - \eta \circ \xi)(0) = 0$

(v) $D\xi(0) = D\eta(0) = D^2\xi(0) = D^2\eta(0) = 0$

(d) $dT_* = dT(\xi_*, \eta_*)$ is a compact operator and the spectrum of dT_* consists of a single real eigenvalue $\varsigma_* \cong -2.83361$ and the rest of the spectrum lies within the disc $\{z : |z| < 0.875\}$.

(e) $\xi_* \circ \eta_* = \eta_* \circ \xi_*$, on a a neighbourhood of 0.

Figure (14) is a schematic diagram which illustrates the probable connection between the two fixed points (ξ_s, η_s) and (ξ_*, η_*) once condition (c)(v) of Table 2.2 is relaxed (although we still require the condition $D\xi, D\eta > 0$ on $\Omega_1 \cap \mathbb{R}$, $\Omega_2 \cap \mathbb{R}$ respectively). There is an extra unstable direction with eigenvalue β_*^{-2} corresponding to the "cross-over" between pairs corresponding to homeomorphisms and those corresponding to non-invertible maps. This extra unstable direction is supposed to extend to the simple fixed point as a stable direction. Indeed, we have not proved the existence of this extra unstable direction (although numerical evidence suggests that it exists) let alone proved the correctness of the geometry suggested by Figure (14).

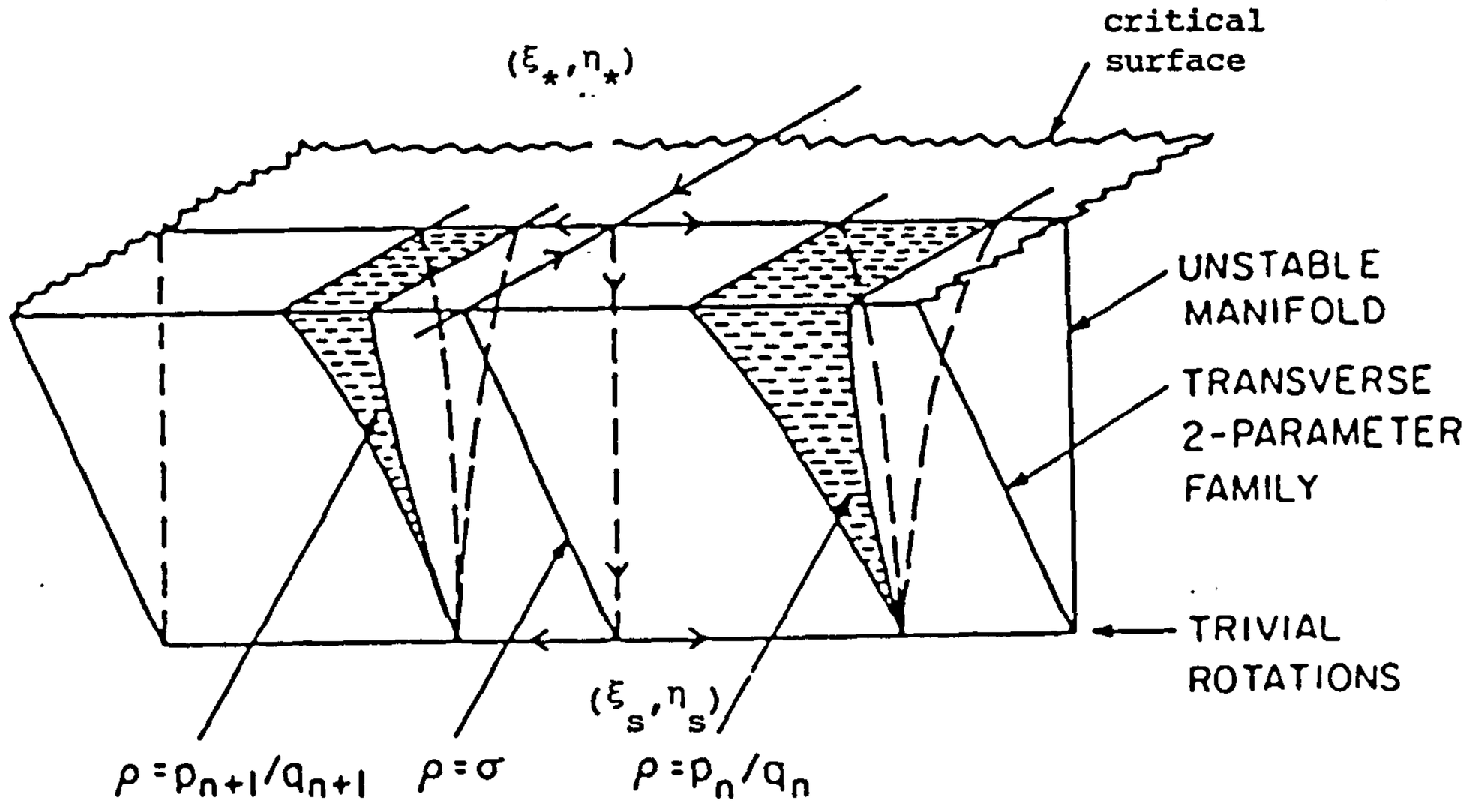


Figure 14. Schematic diagram of the geometry around the fixed points of T. The simple fixed point and the critical fixed point are joined by part of the stable manifold of (ξ_s, η_s) corresponding to the cross-over unstable direction at (ξ_*, η_*).
(Picture taken from Ostlund et al (1983).)

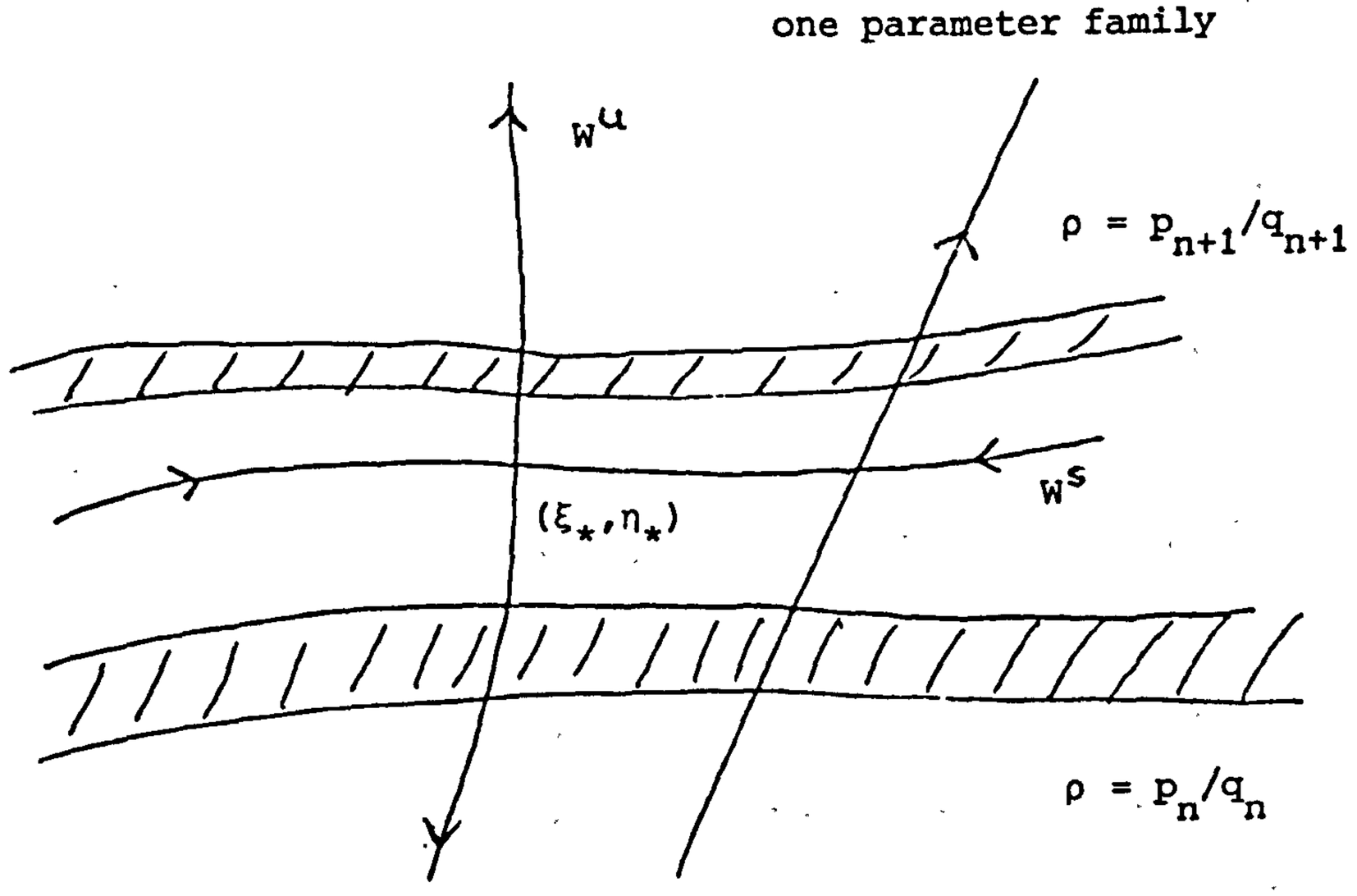


Figure 15. Geometry around the critical fixed point on the critical surface. There is a one dimensional unstable manifold and a co-dimension one stable manifold. A one parameter family that crosses the stable manifold transversely will exhibit universal scaling behaviour.

2.7 Scaling Laws for Cubic Critical Maps

Sections 2.5 and 2.6 details two fixed points of T both of which are hyperbolic with a single simple expanding eigenvalue once we put on constraints to remove extraneous "non-essential" eigenvalues. We may conclude (Appendix 5) the existence of local stable and unstable manifolds for T in the neighbourhood of these fixed points. The stable manifolds W^s are of codimension one and the unstable manifolds W^u are one dimensional.

Let R denote $\{ (\xi, \eta) : \rho(\xi, \eta) = \sigma \}$. We make the following assumptions:

a) R is contained within the stable manifold near to the fixed point.

We know (Lemma 2.2) that all pairs in the stable manifold must lie in R, but as pointed out in (Ostlund et al (1983), section 4.2) there may be parts of R close to the fixed point but not contained in W^s. In Jonker and Rand (1983) it is proved that this assumption is true for the simple fixed point.

Let Σ_n denote the set of pairs (ξ, η) for which $f_{\xi,\eta}$ has 0 as a periodic orbit with rotation number p_n/q_n. Note that

$$T(\Sigma_n) \subseteq \Sigma_{n-1} \tag{2.22}$$

by Lemma 2.2. We assume:

b) W^u intersects Σ_n transversally in a single point for n sufficiently large.

This has been proved for the simple fixed point (Jonker and Rand (1983)). Now let (ξ_μ, η_μ) be a one parameter family of pairs of maps that lie either close to the critical fixed point in the critical manifold (the set of pairs corresponding to circle maps with single cubic critical points) or that lie close to the simple fixed point. (This family may be the embedding of a one parameter family of circle maps f_μ via equation (2.15)) In either case we assume that this family is

transverse to the stable manifold of T. (This behaviour is not unusual for one-parameter families close to the simple fixed point and for families close to the critical fixed point lying in the critical manifold.) Let μ_∞ denote the parameter value at which this transverse interesction takes place. Then for sufficiently large n $(\varepsilon_\mu, \eta_\mu)$ intersects Σ_n in a single point μ_n and

$$\lim_{n\to\infty} \sigma_*^n(\mu_n - \mu_\infty) \qquad (2.23)$$

exists and is non zero. A detailed derivation of this equation from the assumed geometry is contained in Jonker and Rand (1983).

The situation is illustrated in Figure (15). The proof of this result depends upon Proposition A5 taken from Collet et al (1980). This says that it is possible to find a differentiable coordinate change around the fixed point in which the transformation T is linearised in the unstable direction. Geometrically, it is clear that, in view of equation (2.22), the manifolds Σ_n approach the stable manifold W^s at the asymptotic rate σ_*. The fact that the family $(\varepsilon_\mu, \eta_\mu)$ is transverse to W^s means that this geometric rate is transferred to the parameter values μ_n.

2.8 Extensions to Higher Dimensions

Once the renormalisation analysis has been completed for circle maps and circle map pairs, it is possible to extend the theory to higher dimensions and hence to the solutions of differential equations. This is explained partially in Ostlund et al (1983). Further details are given in Rand (1984). We shall outline the basic idea. We embed the one dimensional solution of the renormalisation equations in a higher dimensional space. Let (ε_*, η_*) denote the critical fixed point of Section 2.6. Then we write $\varepsilon_*(x) = h(x^3)$, $\eta_*(x) = k(x^3)$, since (ε_*, η_*) are analytic functions of x^3 (Table 2.2). Let $n > 1$, and choose a unit

vector $\alpha \in \mathbb{R}^{n-1}$. Let (E_*, F_*) denote the pair of maps from $\mathbb{R}^n \cong \mathbb{R} \times \mathbb{R}^{n-1}$ to itself given for $(x,y) \in \mathbb{R} \times \mathbb{R}^{n-1}$ by:

$$E_*(x,y) = (\xi_*(\zeta(x,y)^{1/3}),\ 0) = (h(\zeta(x,y)),\ 0) \qquad (2.24)$$

$$F_*(x,y) = (\eta_*(\zeta(x,y)^{1/3}),\ 0) = (k(\zeta(x,y)),\ 0)$$

where $\zeta(x,y) = X^3 - \langle\alpha,y\rangle$ ("$\langle\cdot,\cdot\rangle$." denotes the usual dot product on $\mathbb{R}^{n-1}$). Here α represents an arbitrary choice of direction in which to embed the one dimensional map. With definition (2.24), (E_*, F_*) is a fixed point of the n dimensional renormalisation transformation S given for pairs of maps (E, F) by:

$$S(E,\ F) = (\Lambda^{-1} \circ F \circ \Lambda,\ \Lambda^{-1} \circ F \circ E \circ \Lambda) \qquad (2.25)$$

where $\Lambda(x,y) = (\beta_* x,\ \beta_*{}^3 y)$.

Then with a suitable space of pairs of functions E, F, S is well defined and C^∞ in a neighbourhood of (E_*, F_*). When the function space is suitably restricted to get rid of extraneous "non-essential" eigenvalues, the derivative $dS_* = dS(E_*, F_*)$ (which is a compact operator) has a spectrum consisting of one simple eigenvalue ς_* and all the rest of the spectrum is contained within the unit disc.

This sets up the renormalisation scheme in the higher dimensional space. Then it is possible to develop the arguments of Section 2.7 directly for higher dimensional systems and so the "ansatz" made by Ostlund et al (1983) that the breakdown of attractive invariant circles may be modelled by one-dimensional circle maps with a single cubic critical point is seen to be unnecessary.

2.9 Experimental Verification

There have now been two experimental attempts to verify that the renormalisation analysis developed here does apply to actual physical systems. One of these (by Libchaber and his coworkers) has not yet been published and we shall only consider the results of

Fein et al (1985).

The results so far are not as conclusive as might be hoped for. I understand that the results of Libchaber are better in this respect. Experimental verification is difficult for a number of reasons. First of all the data from these experiments are time series and power spectra, so it is necessary to understand how to spot universality in these observables. Secondly, we can only expect to see universality when we hold the rotation number or frequency ratio constant at a quadratic irrational value and in particular it is necessary to avoid phase locking. This is a good reason to concentrate on the golden mean as this is the rotation number least likely to be affected by phase locking. Maintaining a constant rotation number requires modulation of the stress parameters discussed in Chapter 1. Fein et al (1985) have performed their experiments on Rayleigh-Bénard Convection. They control the rotation number by modulating the temperature difference.

Making certain assumptions about the power spectrum of cubic critical maps (supported by numerical experiments) Ostlund et al (1983) derive a description of the scaling behaviour of the spectrum predicted by their renormalisation analysis. We shall briefly outline this theory. The power spectrum of a circle map f is defined to be

$$\tilde{f}(\omega) = \lim_{L\to\infty} \sum_{\ell=0}^{L-1} \exp(2\pi i \ell\omega)(f^{\ell}(0) - R_{\sigma}^{\ell}(0)) \qquad (2.26)$$

For cubic critical maps it is conjectured (Ostlund et al (1983), Section 6.2, Conjecture C) that if P is of period 1 and once differentiable (except at 0 where it has left and right derivatives) then:

$$\lim_{L\to\infty} \sum_{\ell=0}^{L-1} \exp(2\pi i \ell\omega) P(f^{\ell}(0)) = O(\omega.\tilde{f}(\omega)) \text{ as } \omega \to 0 \quad (2.27)$$

From this conjecture, they show that for f, g two cubic critical maps

$$\tilde{f}(\omega) - \tilde{g}(\omega) = O(\omega.\tilde{g}(\omega)) \text{ as } \omega \to 0 \quad (2.28)$$

This says that the power spectrum is <u>universal for small ω.</u> Moreover, they predict a scaling law for $\tilde{f}(\omega)$:

$$\tilde{f}(\omega) = -\sigma.\tilde{f}(\sigma^{-1}.\omega) \text{ as } \omega \to 0. \quad (2.29)$$

We should therefore expect that the frequency power spectrum is universal for small frequencies.

Figure (16) shows the power spectrum for the sine map for a = 1 and $\rho = \sigma$. The scales are logarithmic and one can clearly see the self similarity indicating a scaling structure.

Figure (17) shows the corresponding power spectrum obtained from time series measurements on the Rayleigh-Bénard convection cell just above criticality. There is a clear self similarity in the spectral diagram but unfortunately, for very low frequencies the normalisation of the spectrum produces very large background noise.

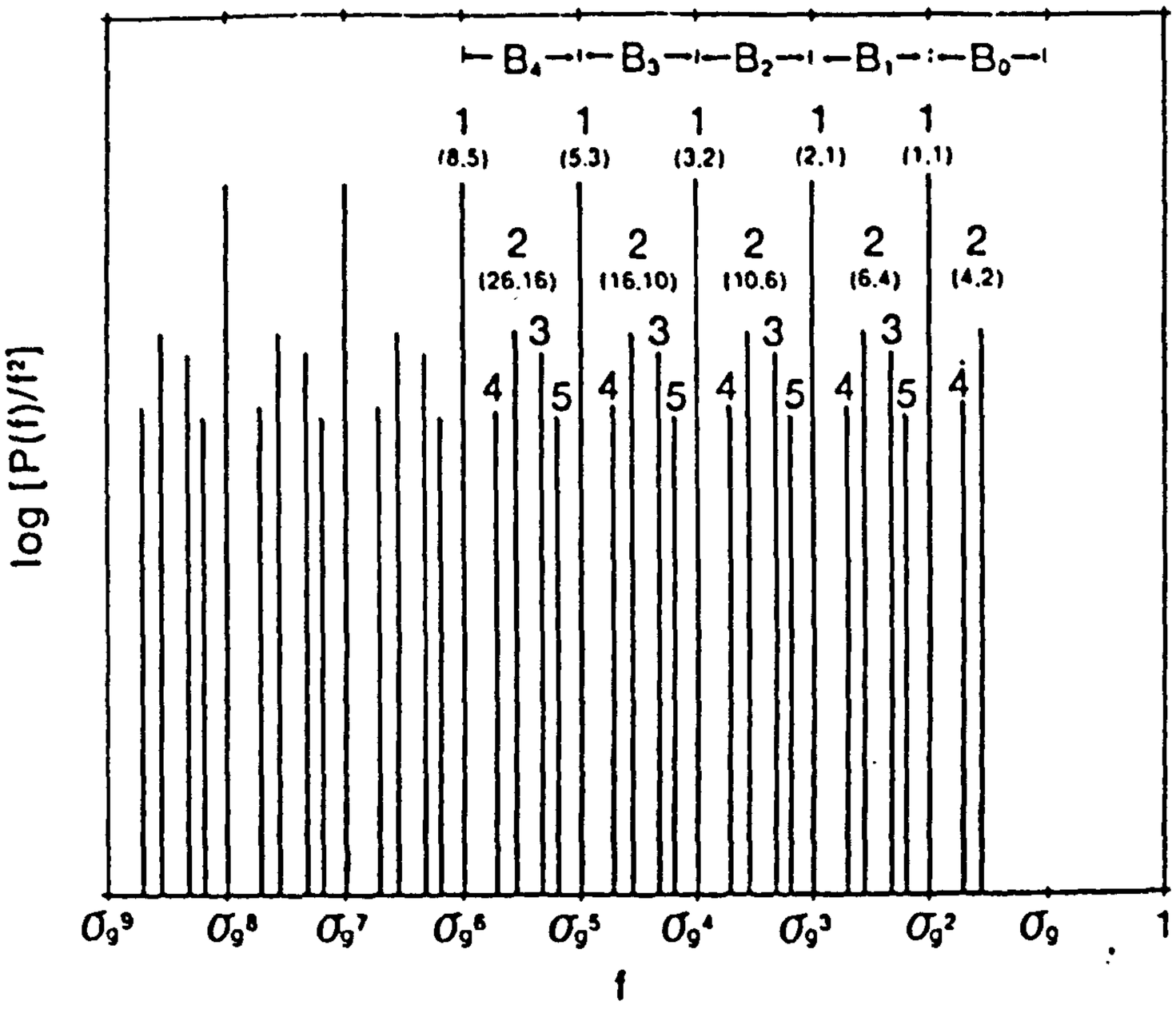


Figure 16. Power spectrum of the sine map for a = 1 and rotation number σ. The principal peaks are at powers of σ and the lines in each band are identical for ω small. (Picture taken from Fein et al (1985).)

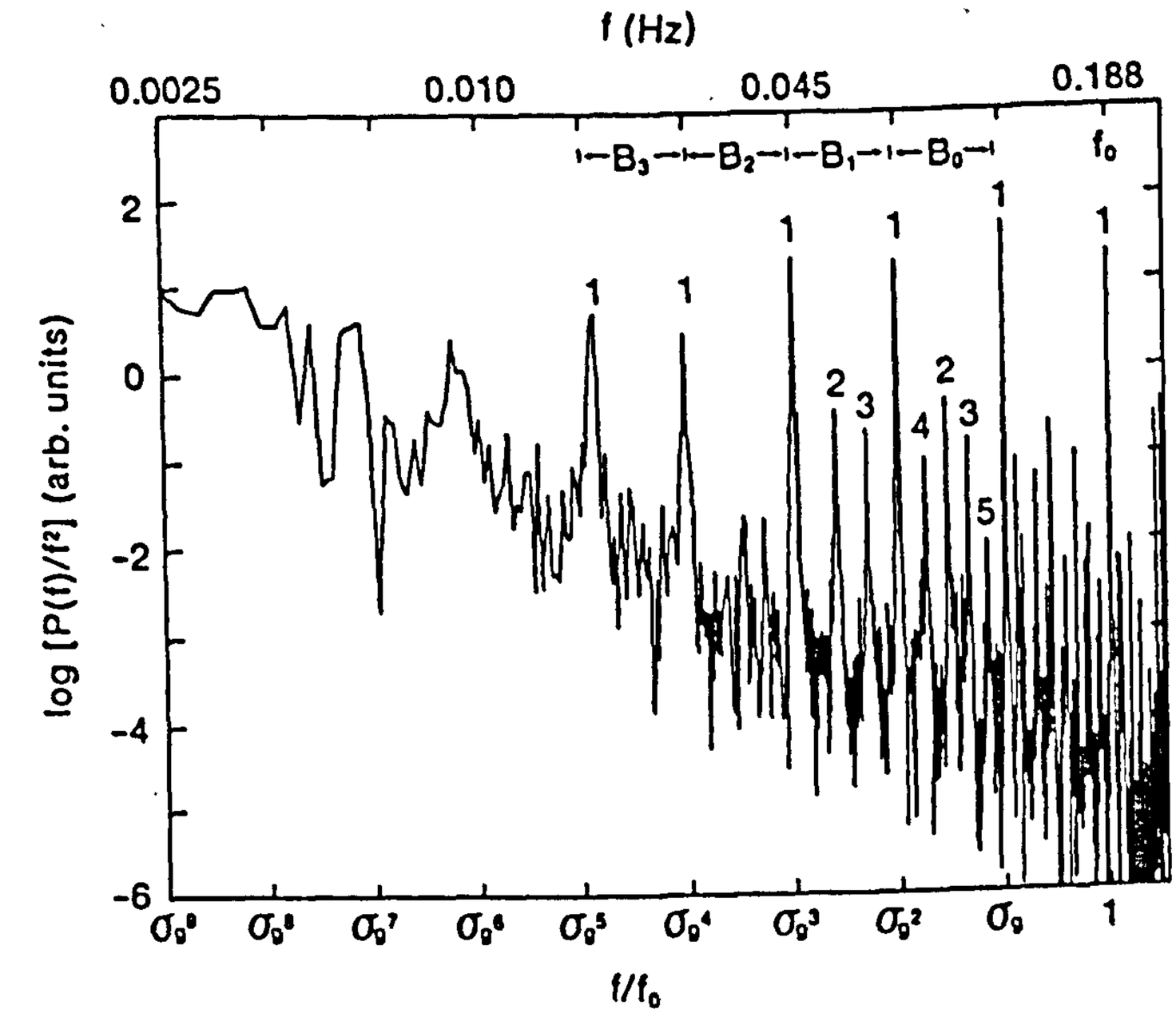


Figure 17. Scaled power spectrum corresponding to quasi-periodic data with golden mean frequency ratio from a Rayleigh-Benard convection cell. (Picture from Fein et al (1985).)

3. Statement and proof of results

In this chapter we state the results proved, give a description of the method of proof, and then explain the parts of the complete proof in detail.

The renormalisation transformation T defined by equation (2.18) has the normalisation condition $\xi(0) = \eta(0) + 1$, which makes the circle of unit length. This is the most natural condition for the embedding of circle maps in the space of pairs of maps as outlined in Chapter 2. However, it is not the most convenient from the computational point of view. Various formulae are simplified if we impose the condition $\xi(0) = 1$ (as in Feigenbaum et al (1982) and McKay (1982)). The expression for β then simplifies to $\beta = \eta(0)$. This merely involves a change of scale. The results of Appendix 4 say that this change makes no difference to the spectrum of the derivative of T. We therefore henceforth define the renormalisation transformation by

$$T(\xi, \eta)(x) = (\beta^{-1}\eta(\beta x), \beta^{-1}\eta(\xi(\beta x))), \quad \beta = \eta(0). \qquad (3.1)$$

3.1 Statement of Results

We first fix some notation. Let

$$a_1 = -63457/2^{18} \cong -0.2420692$$

$$r_1 = 9898557/2^{25} \cong 0.295$$

$$a_2 = 4344147/2^{23} \cong 0.5178627$$

$$r_2 = 9898557/2^{24} \cong 0.590$$

$$\Omega_1 = \{ x \in \mathbb{C} : | x^3 - a_1 | < r_1 \}$$

$$\Omega_2 = \{ x \in \mathbb{C} : | x^3 - a_2 | < r_2 \}$$

We write A for the space $A(\Omega_1) \times A(\Omega_2)$ as defined in Appendix 3. We write A^3 for the subspace of A comprising those pairs $(\xi, \eta) \in A$ such that both ξ and η are analytic functions of x^3. From Proposition

A4.12(a), T preserves A^3. As in Appendix 4, we write for $q \geq 0$, $F^{(q)}(\xi, \eta) = D^q(\xi \circ \eta - \eta \circ \xi)(0)$ and for open sets U, V we write $U < V$ if $cl(U) \subseteq V$.

The main result proved in the computer program is the following:

Theorem 3.1 There is an open set $V^3 \subseteq A^3$ with the following properties:

(1) For all $(\xi, \eta) \in V^3$:

$$\begin{aligned} \beta.\Omega_1 &< \Omega_2 \\ \beta.\Omega_2 &< \Omega_1 \\ \xi(\beta.\Omega_2) &< \Omega_2 \end{aligned} \tag{3.2}$$

where $\beta = \eta(0)$.

(2) $T: V^3 \to A^3$ is well defined and C^∞.

(3) T has a unique fixed point (ξ_*, η_*) in V^3.

(4) For all $(\xi, \eta) \in V^3$, $dT(\xi,\eta) : A^3 \to A^3$ is a compact linear operator.

(5) The spectrum of $dT_* = dT(\xi_*,\eta_*)$ consists of 3 simple eigenvalues:

$$\begin{aligned} s_* = \lambda_0 &\in [-2.84, -2.83] \\ \lambda_1 &\in [2.13, 2.14] \\ \lambda_2 &\in [-1.01, -0.99] \end{aligned} \tag{3.3}$$

and the rest of the spectrum is contained within the disc D(0, 0.875).

(6) The eigenvector $(s\xi, s\eta)$ of dT_* corresponding to λ_1 violates the condition:

$$dF^{(0)}(\xi_*,\eta_*)(s\xi, s\eta) = 0$$

(7) The eigenvector $(s\xi, s\eta)$ of dT_* corresponding to λ_2 violates the condition:

$$dF^{(3)}(\xi_*,\eta_*)(s\xi, s\eta) = 0$$

(8) $\beta_* = \beta(\xi_*,\eta_*) \in [-0.776052, -0.776051]$.

(9) $D^3\xi_*(0), D^3\eta_*(0) > 0$.

(10) $\kappa(\beta_*^2) = \beta_*^2$, $D\kappa(\beta_*^2) < 0$, $\kappa = \beta_*^{-1}.\eta_*$.

(11) $\xi_* \circ \eta_* = \eta_* \circ \xi_*$, on a neighbourhood of 0.

□

We shall prove this Theorem in section 3.3.

We remark that the assumptions (i)-(v) of Section A4.2 hold. For (i) follows from Theorem 3.1 (1) and (3), (ii) follows from Theorem 3.1 (8), (iii) is a simple consequence of the definition of Ω_1, Ω_2, (iv) follows from the fact that ξ_*, η_* are both analytic functions of x^3 together with Theorem 3.1 (9) and, finally, (v) follows from Theorem 3.1 (10).

Let A^{com} be the subset of A for which (ξ, η) satisfy

$$D^i(\xi \circ \eta - \eta \circ \xi)(0) = 0, \text{ for } i = 0, 1, 2, 3. \qquad (3.4)$$

Let A^{com}_{cr} denote the subset of A^{com} for which (ξ, η) satisfy

$$D^i\xi(0) = D^i\eta(0) = 0, \ i = 1,2. \qquad (3.5)$$

From Propositions A4.11 and A4.12, we have that, at least in a neighbourhood of (ξ_*, η_*), A^{com} and A^{com}_{cr} are both manifolds that are preserved by T.

From this, we may deduce the following

<u>Theorem 3.2</u> There is an open set $V \subseteq A$ with the following properties:

(1) For all $(\xi, \eta) \in V$:

$$\beta.\Omega_1 < \Omega_2$$

$$\beta.\Omega_2 < \Omega_1$$

$$\xi(\beta.\Omega_2) < \Omega_2$$

where $\beta = \eta(0)$.

(2) $T:V \to A$ is well defined and C^∞.

(3) T has a unique fixed point (ξ_*, η_*) in V. Both of ξ_*, η_* are analytic functions of x^3.

(4) For all $(\xi, \eta) \in V$, $dT(\xi,\eta) : A \to A$ is a compact linear operator.

(5) The spectrum of $dT_* = dT(\xi_*,\eta_*)$ restricted to A^{com} consists of a

simple single eigenvalue

$$\varsigma_* \in [-2.84, -2.83],$$

together with possibly two other eigenvalues β_*^{-1}, β_*^{-2} and the rest of the spectrum is contained within the disc D(0, 0.875).

(6) The spectrum of $dT_* = dT(\xi_*,\eta_*)$ restricted to A_{cr}^{com} consists of a simple single eigenvalue

$$\varsigma_* \in [-2.84, -2.83],$$

and the rest of the spectrum is contained within the disc D(0, 0.875).

(8) $\beta_* = \beta(\xi_*,\eta_*) \in [-0.776052, -0.776051]$.

(9) $D^3\xi_*(0)$, $D^3\eta_*(0) > 0$.

(10) $\kappa(\beta_*^2) = \beta_*^2$, $D\kappa(\beta_*^2) < 0$, $\kappa = \beta_*^{-1}.\eta_*$.

(11) $\xi_* \circ \eta_* = \eta_* \circ \xi_*$, on a neighbourhood of 0.

□

We remark that the eigenvalue β_*^{-2} is the "cross-over" eigenvalue mentioned in Chapter 2 and illustrated in Figure (14). It corresponds to the addition of a function that is linear at 0. The eigenvalue β_*^{-1} corresponds to the addition of a function that is quadratic at 0. This case is excluded in the analysis of circle maps because it violates the condition (2.15)(iv). The fixed point (ξ_*, η_*) is displayed graphically in Figure (18). From the results of section A4.1, there will be an an analogous theorem for the transformation (2.17). Note that S defined by (A4.17) preserves the conditions (3.4) and (3.5), so that the conclusions of Theorem 3.2 (6), (7) will carry over.

<u>proof of Theorem 3.2 from 3.1</u> The proof of this theorem in a straightforward application of Theorem 3.1, together with the results of Appendix 4. Let (ξ_*, η_*) be the fixed point given by Theorem 3.1. Since equations (3.2) holds for (ξ_*, η_*), they will also hold for $(\xi, \eta) \in A$ close to (ξ_*, η_*). We choose a neighbourhood V of (ξ_*, η_*) in A for whch (3.2) holds and such that $V \cap A^3 \subseteq V^3$ of Theorem 3.1.

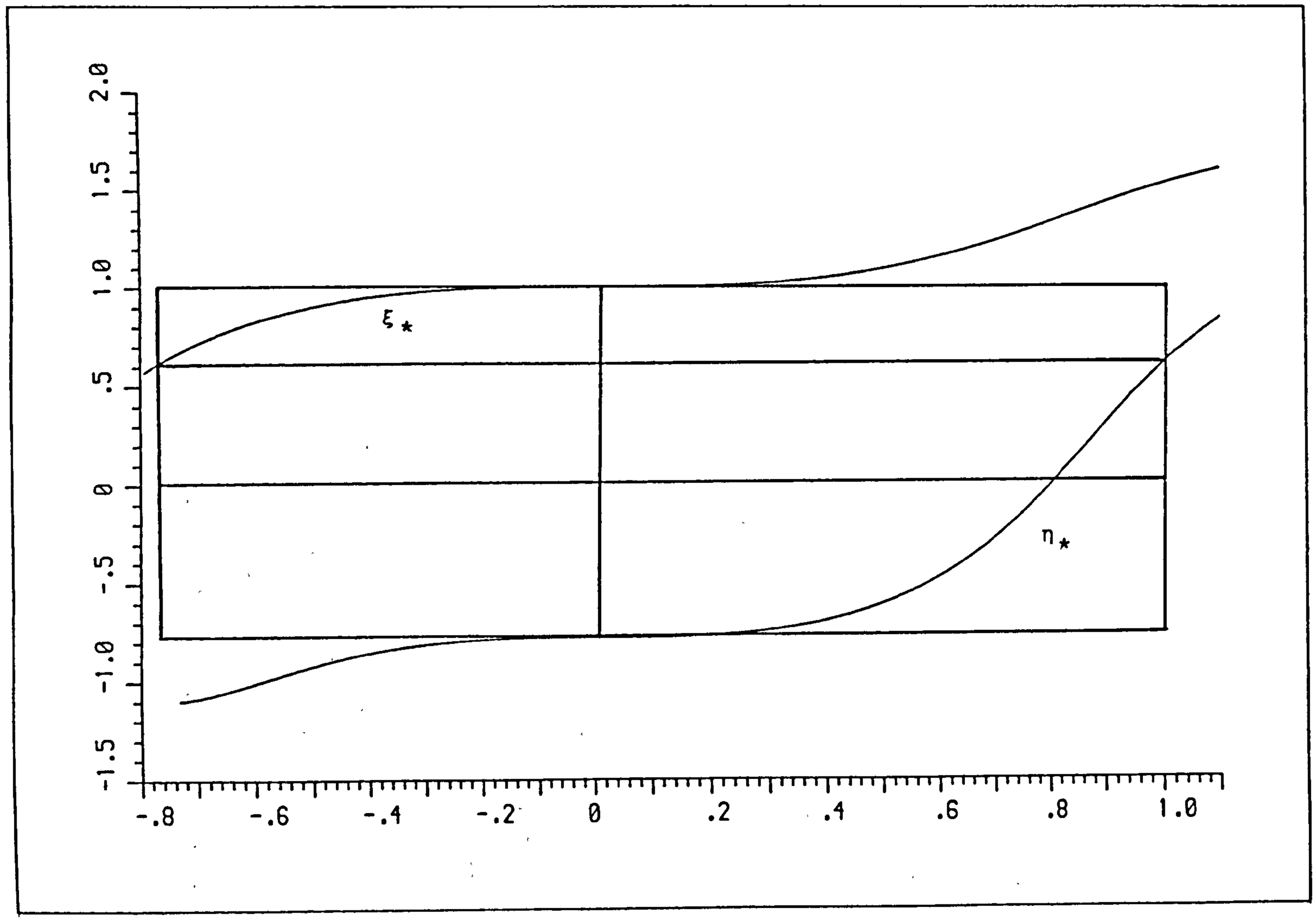


Figure 18. The critical fixed point (ξ_*,η_*) with normalisation $\xi_*(0) = 1$.

That T is well defined and C^∞ follows immediately from (3.2) and Proposition A3.3. This proves (1) and (2). (ξ_*, η_*) is a fixed point in V. The uniqueness is slightly tricky, so we deal with it below. (4) is immediate since $(\xi_*, \eta_*) \in A^3$. (5) follows immediately from (3.2), Proposition A3.6 and the formula for dT (A6.2). The spectrum of dT_* restricted to A^3 is given by Theorem 3.1 (5). Note that Proposition A4.9 says that any eigenvalue of $dT_* = dT(\xi_*,\eta_*)$ on A, which is not an eigenvalue λ of dT_* on A^3, must be of the form $\pm\beta_*^{q-3}$, where q is the smallest integer, not a multiple of three, for which one of $D^q\delta\xi(0)$, $D^q\delta\eta(0)$ is non-zero. Here $(\delta\xi, \delta\eta)$ is an eigenvector of dT_* in A with eigenvalue λ. We note that such a λ does not have modulus one. This means that (ξ_*, η_*) is a hyperbolic fixed point of T. Now by Hartman's theorem (see for example Irwin (1980)) T is conjugate to its linear part in a neighbourhood of (ξ_*, η_*) and thus is an isolated fixed point. Hence, (ξ_*, η_*) is unique provided V is chosen small enough to be included in this neighbourhood. This completes the proof of (3). Now, Proposition A4.13 shows that, if we restrict to A^{com}, then the only possible extra eigenvalues of modulus greater than or equal to one are are those for which $q = 1, 2$, that is, $\lambda = \beta_*^{-2}, \beta_*^{-1}$. Note that Theorem 3.1 (6) and (7) show that the eigenvectors with eigenvalues λ_1, λ_2 violate (3.4). Hence they are not in the spectrum of $dT_*|A^{com}$. (They are in fact $-\beta_*^{-3}$ and -1 respectively - see Ostlund et al (1983).) This proves (5). Restricting further to A_{cr}^{com} removes the possible extra eigenvalues β_*^{-2}, β_*^{-1} and (7) folllows immediately. Finally (8) - (10) are immediate consequences of Theorem 3.1.

□

3.2 Description of the Method of Proof

The proof follows closely the work of Lanford in his proof of the Feigenbaum conjectures. In essence the proof consists of a simple application of the contraction mapping theorem. The following is taken from Krasnoselski et al (1972).

Proposition 3.2 Let e_0 be an element of a real Banach space E. Let V be a neighbourhood of e_0 given by $V = \{e \in E : \|e_0 - e\| < r\}$. Let $N:V \to E$ be a C^1 map, let $\|dN(e)\| \leq \gamma$ for all $e \in V$, and let $\|N(e_0) - e_0\| < \nu$. Then if $c = \gamma + \nu/r < 1$, then N has a unique fixed point in V.

proof Since $\|dN(e)\| < 1$ for all $e \in V$, N is a contraction and the contraction mapping theorem will give a unique fixed point in V provided $cl(N(V)) \subset V$. But for $e \in V$,

$$\begin{aligned}\|N(e) - e_0\| &\leq \|N(e) - N(e_0)\| + \|N(e_0) - e_0\| \\ &\leq \sup\{\|dN(e)\| : e \in V\}.\|e - e_0\| + \|N(e_0) - e_0\| \\ &\leq \gamma.r + \nu \leq (\gamma + \nu/r).r \leq c.r < r\end{aligned}$$

so that $cl(N(V)) \subset V$ and N is a contraction map $V \to V$.

□

The rest of the proof consists of finding E, e_0, V, N and verifying the above estimates! We shall use a generalisation of Newton's method for finding zeroes of non-linear equations to obtain a contraction.

There are several steps in the proof which we describe briefly. The first step is to obtain a good approximate fixed point (ξ_0, η_0) of the renormalisation operator T and good domains on which to define the functions ξ, η. McKay (1982) describes how to use the functional equations and the value of β_* to get a quartic approximation to η_*, expanded about 0. Scaling this approximate fixed point gives an approximation to ξ_*. Application of a Newton or quasi-Newton method

can be used to improve this approximate fixed point by taking more terms. However, to get very good convergence, it is important to find a good domain on which to expand ξ, η. The best domain for η is around the fixed point x_* of the function $\xi_*(\beta_* x)$. An approximate value x_0 of x_* can be found from the expansion of η_0 about 0. Then η_0 is expressed as a Taylor series around this approximate value of x_0 and ξ_0 is expanded about $\beta_0.x_0$, where $\beta_0 = \eta_0(0)$. From this expansion, better ξ_0, η_0 can be obtained (by taking more terms in the Taylor series) and from these, better approximations to x_* and β_*, and so better domains to expand to the functions ξ_0, η_0. In this way the functions ξ_0, η_0 can be improved successively until the functions are accurate to machine precision. We have expanded ξ_0, η_0 up to degree forty. The Taylor coefficients are given in Appendix 7.

After obtaining good domains and good ξ_0, η_0 the next step is to block diagonalise an approximation to the derivative matrix dT. The matrix dT is not close to diagonal and we require a diagonal matrix both to obtain a contraction and also to deduce the spectral properties of the derivative dT at the fixed point (ξ_*, η_*). It is important to take a sufficiently large matrix approximation to guarentee that the error in the approximation is small even when the matrix is transformed under this block diagonalisation. This requires a certain amount of trial and error. The columns of the derivative matrix decrease geometrically, but, unfortunately, at rate ≤ 0.98. This figure is much larger than the rate for the Feigenbaum case and it is therefore necessary to take a considerably larger derivative matrix. This seems to be built into the problem and is the primary cause for the delicacy of the computer calculations. The aim of the

block diagonalisation procces is to transform the derivative matrix so that it has the form:

$$\begin{bmatrix} \lambda_0 & & \\ & \lambda_1 & \\ & & \lambda_2 \\ & & & D \end{bmatrix} \quad \text{where } \lambda_0 \cong -2.83,\ \lambda_1 \cong 2.14,\ \lambda_2 \cong -1, \text{ and } \|D\| < 0.8 \tag{3.6}$$

We are constrained by the fact that by making a change of basis, error in the approximation will be multiplied by the ill-conditioning factor $\|P\|.\|P^{-1}\|$, where P is the matrix transforming the derivative to the above form. Complete diagonalisation of the derivative would make this factor too large and so we use a version of the Jacobi iteration method for non-symmetric matrices (Wilkinson and Reinsch 1971) that converges to the Jordan canonical form through a sequence of complex rotations (i.e. rotations through complex angles) on the basis vectors. We adapted the ALGOL program in Wilkinson and Reinsch (1971) to reduce the number of rotations and stopped the algorithm as soon as the matrix had reached the required form. The rotations were accumulated to form the conjugation matrix P. This basis change was tested to see whether the error remained sufficiently small. We required to take Taylor series up to degree 80 before this method worked satisfactorily. Unfortunately, this method is ad hoc. I do not know of an optimum method to obtain the block diagonalisation that we require.

We now transform to a new basis given by this transformation. Using this new basis it is straightforward to define a ball V around (ξ_0, η_0) and a map N which is a contraction on V. We can then use a method contained in Eckmann et al (1982) to deduce information on the spectrum of dT in V.

Since a fixed point of T is a zero of T - I, a straightforward

generalisation of Newton's method is

$$(\xi, \eta) \to (\xi, \eta) - (dT_{(\xi,\eta)} - I)^{-1}.(T(\xi, \eta) - (\xi, \eta))$$

where the second term on the right hand side is the inverse of the linear operator (DT - 1) applied to the difference $T(\xi, \eta) - (\xi, \eta)$. It is, however, unnecessary to use this formula. For the map

$$(\xi, \eta) \to (\xi, \eta) - J.(T(\xi, \eta) - (\xi, \eta)),$$

where J is an approximation to $(dT - I)^{-1}$ evaluated close to the fixed point, is sufficient to obtain a contraction. In fact, for the spectral calculations, it is important to work in basis in which the derivative matrix is in the form (3.6) above. In this basis, we can take J to be a very simple linear map indeed.

The spectral calculations consist of verifying that, in the ball V, dT has no eigenvalues on various circles in the complex plane. The map dT is shown to be compact by verifying that it is an analyticity improving operator i.e. an operator that produces functions that are analytic on a larger domain (see Appendix 3). The compactness implies that the spectrum of dT consists of discrete eigenvalues λ with finite dimensional eigenspaces (except $\lambda = 0$). Only finitely many λ are outside the unit circle in $\mathbb{C}$. By using perturbation theory we can show that the spectrum of dT has only simple eigenvalues of modulus greater than or equal to one.

Once an open ball containing the fixed point is obtained, then it is straightforward to obtain reasonable estimates on the top three eigenvalues of the derivative at the fixed point (together with their eigenvectors) and to check that λ_1, λ_2 violate (the inifinitesimal versions of) conditions 3.4.

3.3 Proof of Theorem 3.1

The proof uses functional analytic estimates obtained rigorously with the help of a digital computer. The method used in briefly described in Chapter 4. Further details are given in Appendices 7 and 8. The use of the computer is considerably more extensive than in Lanford (1982). We use the computer not only to obtain the estimates, but also to check that these estimates are sufficient to prove the various statements of Theorem 3.1. In this respect I would call the proof a "computer proof" as oppposed to a "computer-assisted proof."

There are two important points to deal with first. Firstly, we note that A^3 is isomorphic to the space $A(a_1, r_1) \times A(a_2, r_2)$ via the isomorphism

$$(\xi, \eta) \longrightarrow (h, k),\ \xi(x) = h(x^3),\ \eta(x) = k(x^3).$$

The renormalisation transformation T induces a transformation (also wriiten as T) on $A(a_1, r_1) \times A(a_2, r_2)$ via this isomorphism. Throughout the program we work with this induced transformation. Appendix 6 lists various formulae both in terms of (ξ, η) and (h, k). We shall, however, state the results in terms of functions (ξ, η).

Secondly, we work in the space $L = L(a_1, r_1) \times L(a_2, r_2)$ rather than $A(a_1, r_1) \times A(a_2, r_2)$ (see Appendix 3 for the definition of these spaces). This is because L^1-norms are readily computable, while the supremum norms are not. (For a discussion of L^1 spaces, see Appendix 3.) The computer program proves the following proposition.

Proposition 3.4 There is an open set $V \subseteq L$ with the following properties:

(1) For all $(h, k) \in V$:

$$\begin{aligned} \beta^3.\Omega_1 &< \Omega_2 \\ \beta^3.\Omega_2 &< \Omega_1 \\ h(\beta^3.\Omega_2)^3 &< \Omega_2 \end{aligned} \tag{3.7}$$

where $\beta = k(0)$.

(2) $T:V \to L$ is well defined and C^∞.

(3) T has a unique fixed point (h_*, k_*) in V.

(4) For all $(h, k) \in V$, $dT(h,k) : L \to L$ is a compact linear operator.

(5) The spectrum of $dT_* = dT(h_*,k_*)$ consists of 3 simple eigenvalues:

$$\delta_* = \lambda_0 \in [-2.84, -2.83]$$

$$\lambda_1 \in [2.13, 2.14]$$

$$\lambda_2 \in [-1.01,-0.99]$$

and the rest of the spectrum is contained within the disc D(0, 0.875).

(6) The eigenvector $(\delta h, \delta k)$ of dT_* corresponding to λ_1 violates the condition:

$$dF^{(0)}(h_*,k_*)(\delta h, \delta k) = 0$$

(7) The eigenvector $(\delta h, \delta k)$ of dT_* corresponding to λ_2 violates the condition:

$$dF^{(3)}(h_*,k_*)(\delta h, \delta k) = 0$$

(8) $\beta_* = \beta(h_*,k_*) \in [-0.776052, -0.776051]$.

(9) $Dh_*(0), Dk_*(0) > 0$.

(10) $g(\beta_*^6) = \beta_*^2$, $Dg(\beta_*^6) < 0$, $g = \beta_*^{-1}.k_*$.

□

Proof of Theorem 3.1 given Proposition 3.4

The key to the proof is that (3.7) will hold for (h, k) in $A(a_1, r_1) \times A(a_2, r_2)$ sufficiently close to (h_*, k_*). We may therefore find a neighbourhood V in $A(a_1, r_1) \times A(a_2, r_2)$ for which (3.7) holds

and hence, translating back to A^3, we may find a neighbourhood V^3 of (ξ_*,η_*) $(\xi_*(x) = h_*(x^3), \eta_*(x) = k_*(x^3))$ for which (3.2) hold. We may further suppose that $V \cap L$ is contained in the neighbourhood V of Proposition 3.4. This proves (1). (2) follows immediately from (3.1), (3.2) and the results of Appendix 3, especially Proposition A3.3. (ξ_*, η_*) is certainly a fixed point of T in V^3. We note that it must be unique. For let (h', k') be a fixed point of T in V. then, in view of (3.7) and (A6.7), (h', k') are defined on domains $D(a_1, r_1')$, $D(a_2, r_2')$ with

$$D(a_1, r_1) < D(a_1, r_1'), \quad D(a_2, r_2) < D(a_2, r_2') \qquad (3.8)$$

From Proposition A3.1 $(h', k') \in A(a_1,r_1') \times A(a_2,r_2') \subseteq L(a_1, r_1) \times L(a_2, r_2)$ so that (h', k') are in the neighbourhood V of Proposition 3.4 so that by (3) of that Proposition $(h', k') = (h_*, k_*)$. (4) follows from the relations (3.2) and Proposition A3.6.

Now let $(\delta h, \delta k)$ be an eigenvector of $dT_* = dT(h_*, k_*)$ with non-zero eigenvalue λ. $(\delta h, \delta k)$ are defined on $D(a_1,r_1')$, $D(a_2, r_2')$ respectively in view of (3.7) and (A6.8). Thus $(\delta h, \delta h) \in L$ and hence any eigenvector of dT_* in $A(a_1, r_1) \times A(a_2, r_2)$ also lies in L. The converse is trivially true by Proposition A3.1(c). This proves (5). (6) and (7) are direct translations of Proposition 3.4 (6) and (7). Similarly (8), (9) and (10) are direct translations of the statements (8), (9), (10) of Proposition 3.4. (11) follows from Proposition A4.6, noting that assumptions (i) - (v) of Section A4.2 are satisfied by (10), (11), (12), the fact that ξ_*, η_* are analytic functions of x^3, and that $0 \in \Omega_1, \Omega_2$, and $[0, 1] \subseteq \Omega_2$.

□

We now turn to the proof of Proposition 3.4. The proposition is proved with the aid of the computer program circ_proof in Appendix

7. The space L (with norm $\|(h, k)\| = \| h \| + \| k \|$) is isometrically isomorphic to the space $\ell_1 \oplus \ell_1$ (see Appendix 3 for a definition of ℓ_1). This isomorphism is given by the basis $\{(e_i^{(1)}, 0) : i \geq 0\} \cup \{(0, e_i^{(2)}) : i \geq 0\}$ where

$$e_i^{(1)} = ((x - a_1)/r_1)^i, \quad e_i^{(2)} = ((x - a_2)/r_2)^i$$

for $i = 0, 1, \ldots$ We shall refer to this basis on L as the standard basis of L. Dividing the standard basis into

$$\{ (e_i^{(1)}, 0) : 0 \leq i \leq 80 \} \cup \{ (e_i^{(1)}, 0) : 81 \leq i \} \cup$$

$$\{ (0, e_i^{(2)}) : 0 \leq i \leq 80 \} \cup \{ (0, e_i^{(2)}) : 81 \leq i \}$$

we may look on L as $\mathbb{R}^{81} \oplus \ell_1 \oplus \mathbb{R}^{81} \oplus \ell_1$.

We consider the operator matrix

$$\begin{bmatrix} P_{11} & 0 & P_{12} & 0 \\ 0 & I & 0 & 0 \\ P_{21} & 0 & P_{22} & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \in \Lambda(\mathbb{R}^{81} \oplus \ell_1 \oplus \mathbb{R}^{81} \ell_1, \; \mathbb{R}^{81} \oplus \ell_1 \oplus \mathbb{R}^{81} \ell_1)$$

where for Banach spaces E, F, $\Lambda(E, F)$ denotes the space of bounded linear maps of E -> F. Here I is the identity operator on ℓ_1. We shall assume that the 164 × 164 matrix

$$\begin{bmatrix} P_{11} & 0 & P_{12} & 0 \\ 0 & 1 & 0 & 0 \\ P_{21} & 0 & P_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \Lambda(\mathbb{R}^{164}, \mathbb{R}^{164})$$

is an invertible matrix, with inverse

$$\begin{bmatrix} P_{11}^{-1} & 0 & P_{12}^{-1} & 0 \\ 0 & 1 & 0 & 0 \\ P_{21}^{-1} & 0 & P_{22}^{-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \Lambda(\mathbb{R}^{164}, \mathbb{R}^{164})$$

Then P is invertible with inverse

$$\begin{bmatrix} P_{11}^{-1} & 0 & P_{12}^{-1} & 0 \\ 0 & I & 0 & 0 \\ P_{21}^{-1} & 0 & P_{22}^{-1} & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \quad \epsilon \Lambda(\mathbb{R}^{81} \oplus \ell_1 \oplus \mathbb{R}^{81} \oplus \ell_1, \mathbb{R}^{81} \oplus \ell_1 \oplus \mathbb{R}^{81} \oplus \ell_1)$$

The basis

$$f_i^{(1)} = (e_i^{(1)}, 0)P, \quad f_i^{(2)} = (0, e_i^{(2)})P$$

for $i = 0, 1, 2, \ldots$ is a basis for ℓ_1 which we call the p-basis. (The left multiplication is because P is really a matrix on the coordinates rather than the basis vectors themselves.) The basis change matrix P is calculated as explained in Section 3.2.

Theoretically, we shall work with the p-basis. We shall look on the space L as $\mathbb{R}^3 \oplus \ell_1$ regarding the three basis vectors $f_0^{(1)}$, $f_1^{(1)}$, $f_2^{(1)}$ as $\mathbb{R}^3$ and the rest of the p-basis as constituting ℓ_1. The basis vectors $f_0^{(1)}$, $f_1^{(1)}$, $f_2^{(1)}$ are (very) approximately the eigenvectors of dT_* corresponding to the three eigenvalues λ_0, λ_1, λ_2 of modulus greater than or equal to one. The p-basis provides a new L^1-norm for L which we shall use to define balls around the approximate fixed point (see Appendix 3, Section A3.5). The approximate Newton map N is very easy to define with respect to the p-basis:

$$N : I - J(T - I)$$

where J is the diagonal matrix

$$\begin{bmatrix} -1/3.234 & & & \\ & 1/1.239 & & \\ & & -1/2 & \\ & & & -I \end{bmatrix} \quad \epsilon \Lambda(\mathbb{R}^3 \oplus \ell_1, \mathbb{R}^3 \oplus \ell_1)$$

or, more precisely, a binary approximation to this matrix. The program takes an approximate fixed point (h_0, k_0) that is extremely accurate. It defines a ball V or radius r (in terms of the L^1-norm with respect to the p-basis)

$$r \cong 5 \times 10^{-12}$$

and checks directly that T is well defined and C^∞ on V. This is done by checking that (3.7) holds. It then calculates upper bounds for

$$\|N(h_0, k_0) - (h_0, k_0)\|$$

and

$$\sup \{ \|dN(h, k)\| : (h, k) \in V \}.$$

It then checks directly that the conditions of Proposition 3.3 are satisfied for $e_0 = (h_0, k_0)$. We may then conclude the existence of a unique fixed point (h_*, k_*) in V. The derivative $dT_* = dT_*(h_*, k_*)$ may be looked upon as the matrix of operators

$$\begin{bmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \beta_0 \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \beta_1 \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \beta_2 \\ \gamma_0 & \gamma_1 & \gamma_2 & \varsigma \end{bmatrix} \in \Lambda(\mathbb{R}^3 \oplus \ell_1, \mathbb{R}^3 \oplus \ell_1)$$

where for i, j = 0, 1, 2

$$\alpha_{ij} \in \mathbb{R}, \quad \beta_i \in \Lambda(\ell_1, \mathbb{R}) = \ell_1^*, \ \gamma_j \in \Lambda(\mathbb{R}, \ell_1) \cong \ell_1$$

$$\varsigma \in \Lambda(\ell_1, \ell_1)$$

The program calculates upper and lower bounds for α_{ij} and upper bounds for the norms $\|\beta_i\|$, $\|\gamma_j\|$, $\|\varsigma\|$. By a method of Eckmann et al (1982) (described in Appendix 7), the program shows that for $t \in [0, 1]$ any operator of the form

$$L_t = \begin{bmatrix} \alpha_{00} & t\alpha_{01} & t\alpha_{02} & t\beta_0 \\ t\alpha_{10} & \alpha_{11} & t\alpha_{12} & t\beta_1 \\ t\alpha_{20} & t\alpha_{21} & \alpha_{22} & t\beta_2 \\ t\gamma_0 & t\gamma_1 & t\gamma_2 & \varsigma \end{bmatrix} \in \Lambda(\mathbb{R}^3 \oplus \ell_1, \mathbb{R}^3 \oplus \ell_1)$$

has no eigenvalues on the circles

$$\Gamma_1 = C(-2\ 1441151880758587/2^{54},\ 57646075230342349/2^{59})$$

$$\Gamma_2 = C(-1,\ 57646075230342349/2^{59})$$

$$\Gamma_3 = C(0,\ 7/2^3)$$

$$\Gamma_4 = C(2\ 1261007895663739/2^{53},\ 57646075230342349/2^{59})$$

The family $t \to L_t$ is a continuous one parameter family of compact operators on $\mathbb{R}^3 \oplus \ell_1$. Note that

$$L_0 = \begin{bmatrix} \alpha_{00} & & & \\ & \alpha_{11} & & \\ & & \alpha_{22} & \\ & & & \varepsilon \end{bmatrix} \in \Lambda(\mathbb{R}^3 \oplus \ell_1,\ \mathbb{R}^3 \oplus \ell_1)$$

and $L_1 = dT_*$. Now, L_0 has one simple eigenvalue inside each of Γ_1, Γ_2, Γ_4 with the rest of the spectrum contained within Γ_3. Now from, Proposition A3.8, this is true for all L_t with $t \in [0, 1]$ and so, in particular, is true for $L_1 = dT_*$. Thus dT_* has spectrum consisting of three simple eigenvalues λ_0, λ_1, λ_2 contained within Γ_1, Γ_2, Γ_4 respectively with the rest of the spectrum lying within Γ_3. The bounds for the eigenvalues given in Proposition 3.4 are obtained below. This proves (1), (20), (3), (4) of Proposition 3.4. The program obtains (8) and (9) by direct calculation.

The next stage is to obtain bounds for the eigenvalues λ_0, λ_1, λ_2 and their eigenvectors. For each $\ell = 0, 1, 2$ the following is done. First of all, we obtain (numerically) approximate values λ', u' for the eigenvalue λ_* and and its eigenvector u_*. With respect to the p-basis

$$\lambda_* \cong \alpha_{\ell\ell}, \text{ and } u_* \cong k.f_\ell^{(1)}, \text{ for some } k \in \mathbb{R}.$$

In order to obtain a space with a unique eigenvector, we must normalise the eigenvector. We choose to fix u_ℓ, the ℓth component of the vector u, to be the ℓth component of u', where u, u' are expressed in terms of the p-basis. We then consider the pair (λ, u) as lying in the ℓ_1 via the embedding

$$(\lambda, u) = (u_0, \lambda, u_2, \ldots)$$

where the term u_ℓ has been replaced by λ in this expression. We now consider the map

$$S(\lambda, u) = (dT_* - \lambda I) u \qquad (3.4)$$

Then S is map from ℓ_1 to itself. We again look on ℓ_1 as $\mathbb{R}^3 \oplus \ell_1$ via the p-basis. A zero of this map corresponds to an eigenvalue λ_* with eigenvector u_*. We now follow the same procedure for the pair (λ,u) as we did for the pair (h, k). We obtain a contraction on a ball about the approximate eigenvalue/vector by forming an approximate Newton map N_i defined by

$$N_i = I - M.S$$

where M is the diagonal matrix $\cong S^{-1}$ given in its decimal approximation by

$$M = \begin{bmatrix} (\alpha_{0\bar{0}} - \lambda')^{-1} & & & \\ & (u'_{\ell})^{-1} & & \\ & & (\alpha_{22} - \lambda')^{-1} & \\ & & & -\lambda'^{-1} \end{bmatrix} \quad \in \Lambda(\mathbb{R}^3 \oplus \ell_1, \mathbb{R}^3 \oplus \ell_1)$$

Here the ℓth diagonal element is as shown. The program now once again calculates the norm $\|N_i(\lambda',u') - (\lambda',u')\|$ and forms a ball V_ℓ around (λ' ,u') and calculates

$$\sup \{ \| dN_i(\lambda, u) \| : (\lambda, u) \in V_\ell \}.$$

The program then checks the conditions of Proposition 3.3 and, provided all is well, we may conclude that there is a unique eigenvector u_* in V_ℓ. The radius of V_ℓ provides the bounds on the eigenvector given in (5). For i = 1, 2, using the ball around u', the program checks that any u vector in V_i fails to satisfy the condition

$$dF^{(0)}(h_*, k_*)u = 0 \text{ for } i = 1 \text{ and}$$

$$dF^{(3)}(h_*, k_*)u = 0 \text{ for } i = 2.$$

This proves (6) and (7).

□

4. Interval Arithmetic and Numerical Functional Analysis

4.1 Interval Arithmetic

With the advent of fast digital computers, there has been a corresponding upsurge in research into the propogation of error in arithmetic processes. Interval arithmetic was invented in the 1960's as a method of obtaining exact error bounds for the results of numerical calculations. Its use was envisaged primarily for situations in which there was uncertainty in the input data, such as when the data arose from inexact physical measurements. I think that it is fair to say that the usefulness of interval arithmetic in this area has proved to be limited. The error bounds obtained are often considerable overestimates. This is essentially because interval arithmetic treats every step in a sequence of calculations as independent. However, interval arithmetic has proved to be amply adequate for our purposes, once its limitations have been recognised and guarded against. Its use combined with careful analysis of computer rounding error has turned suggestive computer experiment into rigorous mathematical proof. A good introduction to interval arithmetic is Moore (1966).

The fundamental idea of interval arithmetic is to define arithmetic operations on finite closed intervals in $\mathbb{R}$ that correspond to the usual arithmetic operations on $\mathbb{R}$. Suppose that $\odot$ represents a binary operation on $\mathbb{R}$; $\odot$ could be addition or multiplication for example. Then if I_1 and I_2 are two intervals of $\mathbb{R}$, a natural requirement for any definition of an interval $I_3 = I_1 \odot I_2$ is:

$$I_3 = I_1 \odot I_2 \supseteq \{ x_1 \odot x_2 : x_1 \in I_1,\ x_2 \in I_2 \} \qquad (4.1)$$

Of course, ideally, equality would hold. However, for computer

implememented operations, this will not in general be true and indeed it is unnecesary for our purposes. It is important, of course, to make this difference as small as possible. An inclusion similar to (4.1) should hold for unary operations, such as negation. We define interval arithmetic routines to correspond to addition, subtraction, negation, multiplication, inversion and division. Appendix 8 contains a detailed description of the standard interval arithmetic operations and their computer implementation. As two simple examples of interval arithmetic operations we consider addition and multiplication.

Let $I_1 = [a_1,b_1]$, $I_2 = [a_2,b_2]$ be two intervals.

(i) addition Define $I_1 + I_2 = I_3$, where $I_3 = [a_1 + a_2, b_1 + b_2]$. Then this interval addition satisfies (4.1) with the inclusion replaced by equality.

(ii) multiplication Define $I_1 \times I_2 = I_3$, where I_3 is the interval $[a_3, b_3]$ and

$$a_3 = \min \{ a_1.a_2, a_1.b_2, b_1.a_2, b_1.b_2 \}$$

$$b_3 = \max \{ a_1.a_2, a_1.b_2, b_1.a_2, b_1.b_2 \} .$$

This definition again satisfjes (4.1) with equality.

These two interval operations satisfy some but not all of the usual axioms of R. For example, addition and multiplication are both commutative and associative and there are zero ([0, 0]) and unit ([1, 1]) elements. The distributive law does not hold as can be seen from the following example:

$$[-1, 1] \times ([-1, -1] + [1, 1]) = [0, 0]$$

while

$$([-1, 1] \times [-1, -1]) + ([-1, 1] \times [1, 1]) = [-2, 2].$$

However, the following law does hold:

$$I_1 \times (I_2 + I_3) \subseteq (I_1 \times I_2) + (I_1 \times I_3) \qquad (4.2)$$

This means that care has to be taken to "take out common factors"

whenever possible during calculations. Of course, the reason for the problem is that I_1 appears twice on the right hand side of (4.2), and interval arithmetic regards these two intervals as independent of each other. When implemented on a computer these laws will not hold exactly. This is because computer multiplication and addition is in general non-commutative and non-associative. They will, however, be approximately true.

We conclude this section with a definition that will be used often in the Appendices.

Definition An interval of the form $[-a_1, a_1]$ is called a symmetric interval.

4.2 Computer implementation of Interval Arithmetic

The standard interval arithmetic operations can be adapted for implementation on a computer by enlarging the interval computed using standard interval arithmetic to take into account the rounding error made by the computer. This is described in detail in Appendix 8. However the general idea is as follows. If we let R be the set of numbers that can be represented on the computer, then, for $x, y \in R$, the computer-calculated result for the sum of x and y, $x +_c y$, will, in general, not be exact. However, for $r \in R$, there are two functions r_up(r), r_down(r) which give numbers slightly greater and less than r respectively. These functions are such that for all $x, y \in R$

$$r\_down(x +_c y) \leqslant x + y \leqslant r\_up(x +_c y).$$

A similar set of inequalities holds for multiplication. Now it is possible to define an addition operation between those intervals with end-points in R. For example the interval addition defined earlier is modified as follows. Let $I_1 = [a_1,b_1]$, $I_2 = [a_2,b_2]$ be two such

intervals. Then defining and interval addition $+_c$ for the computer by:

$$I_1 +_c I_2 = [r\_down(a_1 + a_2),\ r\_up(b_1 + b_2)]$$

then equation (4.1) will hold for 0 set to $+_c$, although equality will not hold. In this way the standard interval arithmetic operations are implemented on the computer.

4.3 Numerical Functional Analysis

We have used the technique for Numerical Functional Analysis that is described in Eckmann et al (1982). They develop a series of operations on sets of analytic functions, which we call "function balls," that can be implemented as finite algorithms on a digital computer. These algorithms use the interval arithmetic routines that are described in the previous two sections.

For simplicity, we will assume that all functions are real analytic and defined on the unit disc $D(0, 1)$ in $\mathbf{C}$. We shall represent functions by their Taylor expansions about 0. The formulae can be readily extended to any disc in $\mathbf{C}$. Of course, we shall have to truncate these expansions after a finite number of terms and lump anything remaining in an extra term. We therefore choose a positive integer n which will be the degree of the Taylor Expansion. This n will remain fixed throughout. For

$$f(x) = f_0 + f_1.x + f_2.x^2 + \ldots$$

we write $\| f \|$ for the L^1-norm of f given by

$$\| f \| = |f_0| + |f_1| + |f_2| + \ldots$$

We work in the space $L(0, 1)$. This is a space of functions analytic on $D(0, 1)$ with finite L^1-norm. This is formally described in Appendix 3. It is natural to consider sets of functions of the form

$\{ f_p(x) + f_h(x) : \| f_h \| < \epsilon \}$ where f_p is (fixed) polynomial of degree $n > 0$ and f_h is an arbitrary function containing only terms $O(x^{n+1})$ and with L^1-norm bounded by $\epsilon > 0$. Of course, we shall have to replace the Taylor coefficients with intervals. However, we shall also need to introduce an extra arbitrary function, an error function f_e. This is because when we take a function f_h, with only terms that are $O(x^{n+1})$ and compose it with a function that has a non-zero constant term, then the resulting function is made up of terms of all degrees. This function must be taken into account in some form of error term.

Following Eckmann et al (1982) we define a <u>vector</u> v as a finite data structure consisting of (n+1) intervals $v_0, v_1, \ldots, v_n$, together with two non-negative numbers v_h and v_e. As in Eckmann et al (1982), we write a vector as

$$v = \{ v_0, v_1, v_2, \ldots, v_n; v_h, v_e \}.$$

Provided the end-points of the intervals and the numbers v_h, v_e are all in R, the set of numbers that can be represented on the computer, then this data structure can be implemented on a computer. To a vector v we associate a <u>(function) ball</u> B(v) of functions on D as follows:

$$B(v) = \left\{ \begin{array}{l} f_p(x) + f_h(x) + f_e(x) \text{ with} \\ f_p(x) = \sum_{i=0}^{n} f_i.x^i, \quad f_i \in v_i \\ \| f_h \| \leqslant v_h, \ \| f_e \| \leqslant v_e \\ f_h \text{ has only terms } O(x^{n+1}) \end{array} \right\}.$$

We refer to the functions f_h, f_e as the <u>high order</u> and <u>error</u> functions respectively. As with intervals, for many binary or unary operations on functions, we can define operations between vectors to correspond to these operations on the function balls. Suppose that $\odot$

is a binary operation on functions. Then analogously to (4.1) we shall require that for any vectors v and w, v ⊙ w satisfies:

$$\{ f \odot g : f \in B(v),\ g \in B(w) \} \subseteq B(v \odot w) \qquad (4.3)$$

A similar inclusion will hold for unary operators. Again, this inclusion will be strict when implemented on the computer. In Appendix 8 the operations that we have implemented are listed. They include addition, negation and subtraction, multiplication by a "scalar" real number, multiplication (but <u>not</u> division) between functions and, importantly, function composition and differentiation followed by composition.

To illustrate this procedure we give two examples: (i) function addition and (ii) differentiation followed by composition.

Let w, v be two vectors and let

$$f = f_p + f_h + f_e \in B(v), \quad g = g_p + g_h + g_e \in B(w).$$

(i) <u>Addition</u>

We require a definition of addition between v and w so that (4.3) is satisfied. Now

$$f + g = (f_p + g_p) + (f_h + g_h) + (f_e + g_e),$$

where $f_p + g_p$ is a polynomial of degree n and $f_h + g_h$ contains only terms $O(x^{n+1})$. Furthermore,

$$f_p + g_p = (f_0 + g_0) + (f_1 + g_1).x + (f_2 + g_2).x^2 + \ldots$$

and

$$\| f_h + g_h \| \leq \| f_h \| + \| g_h \|, \ \| f_e + g_e \| \leq \| f_e \| + \| g_e \|$$

Therefore, if $u = v \odot w$ is defined by:

$$u_i = v_i + w_i,\ i = 0, \ldots, n \text{ and } u_h = v_h + w_h,\ u_e = v_e + w_e$$

then (4.3) is easily seen to hold.

(ii) <u>Differentiation followed by composition</u>

This example, while rather complicated, is highly instructive. Consider a function f with $\| f \| \leq C < \infty$. Unfortunately, without further

information about f, it is impossible to find a bound for || Df ||. This is why it is not possible to define a straight diferentiation operation. However if g is a function with || g || $\leq$ c < 1, then g(D) $\subseteq$ D and Df o g is well defined. Moreover, || Df o g || can be bounded in terms of c. For if

$$f(x) = f_0 + f_1.x + f_2.x^2 + \ldots$$

then

$$\| Df \circ g \| \leq |f_1| + 2.|f_2|.\| g \| + 3.|f_3|.\| g \|^2 + \ldots$$

$$\leq \| f \|.\ \sup \{ (i + 1).c^i : i > 0\}$$

This last supremum can be calculated in finite time since $(i + 1).c^i$ decreases for $i > a/(1 - a)$. Now for $f \in B(v)$ and $g \in B(w)$ we have Df o g is given by

$$Df_p \circ g + Df_h \circ g + Df_e \circ g$$

We can therefore treat the elements Df_p o g, Df_h o g and Df_e o g independently and add the results together at the end. This is the same as treating v as the sum of the vectors $v_P = \{ v_0, \ldots, v_n; 0, 0 \}$, $v_H = \{ [0,0], \ldots, [0,0]; v_h, 0 \}$, and $v_E = \{ [0,0], \ldots, [0,0]; 0, v_e \}$.

The first of these can be readily defined directly in terms of the composition operation (see appendix 8). Since Df_p has Taylor expansion

$$f_1 + 2.f_2.x + 3.f_3.x^2 + \ldots\ n.f_n.x^{n-1}$$

we see that an appropriate definition for v_P is u_P o w, where u_P is the vector $\{ v_1, 2.v_2, \ldots, n.v_n, [0,0]; 0, 0 \}$. Here $i.v_i$ denotes scalar multiplication of the interval v_i by the integer i. The appropriate definitions for v_H and v_E are clear from the discussion above. If we define u_H and u_E by the vectors:

$\{ [0,0], \ldots, [0,0]; 0; u_h \}$ and

$\{ [0,0], \ldots, [0,0]; 0; u_e \}$ where

$u_h = \sup \{ (i+1).c^i : i \geq n \}$ and

$u_e = \sup \{ (i+1).c^i : i \geq 0 \}$, $\| g \| \leq c < 1$

then $u = u_P + u_H + u_E$ is a definition of differentiation and composition that will satisfy (4.3).

The same general comments about the importance of taking out common factors etc... apply just as well to operations with vectors as to interval arithmetic.

Using computer interval arithmetic, these operations can all be implemented as finite algorithms on the computer. In Appendix 8, there is a complete list of the operations between functions that have been implemented as operations on vectors.

Apart from being necessary to take account of computer rounding error, there are other advantages to this method of numerical functional analysis. First of all, we need to work with a neighbourhood of an approximate fixed point and this "vector arithmetic" provides a natural framework for this to be done. Furthermore, we need to make estimates on the error in truncating the derivative matrix at a finite point. This can be done by evaluating the derivative at all but finitely many basis elements at one go. For the ball $B(\{ [0,0], \ldots, [0,0]; 1; 0 \})$ includes all the basis elements $\{ x^{n+1}, x^{n+2}, \ldots \}$ that we use in the calculation of the derivative matrix.

4.4 Summary

We see that by a careful analysis of the computer algorithms to obtain rigorous error bounds on computer floating point calculations, together with the interval arithmetic and numerical functional

analysis described here, it is possible to rigorous calculations on function balls.

5. Philosophical and Mathematical Considerations

Computer assisted proofs are no longer the novelty that they were ten years ago. The proof of the Four Colour Theorem announced by Appel and Haken (1976) generated a great deal of discussion on the philosophical implications of such proofs. We shall outline some of this discussion below. However, since that time, computers have been used extensively to prove/disprove conjectures in a number of branches of mathematics especially number theory and algebra. By and large, this use of computers has been accepted by the mathematical community, although it has not been universally welcomed. It was thus a natural progression for Lanford to extend the use of computers from combinatorial and integer arithmetical calculations to the rigorous floating point calculations of numerical functional analysis. This extension adds little new to the philosophical questions. However it raises a different and, in my view, more important mathematical question. Even if one accepts these type of computer proofs as valid, it is debatable whether they constitute "good mathematics." Lanford (1981) himself, referring to the Feigenbaum problem, expressed his dissatisfaction as follows:

"... the proofs rest on long and relatively blind computations which could perfectly well, so far as one can see without actually doing them, have come out differently. I think it is fair to say that, although we know <u>that</u> a solution (of the Feigenbaum equation) exists, we don't at all understand <u>why</u> it must exist. In view of the simplicity of the functional equation, this seems a most unsatisfactory state of affairs."

We shall discuss briefly this and other questions below.

5.1 The Controversy over the Four Colour Theorem

When Appel, Haken and Koch (1976) announced their proof of the Four Colour Theorem (4CT) almost ten years ago, it generated a great deal of excitement in the mathematical world. The theorem states that every planar graph may be coloured with four differrent colours so that no adjoining vertices are coloured the same. This theorem, which had eluded proof for so many years, had finally been cracked and then only by extensive use of computer calculations. However, it was not long before philosophers stepped in to spoil the fun. Was the 4CT a theorem at all? Or, rather, did the proof of the 4CT necessitate a revision of the notion of a mathematical theorem? Futhermore, did the proof of the 4CT pose a challenge to the widely held belief that mathematical theorems are a priori truth, existing independently of the physical world? In order to get the flavour of the controversy, we reproduce some the arguments that Tymoczko presented in an article in the Journal of Philosophy together with two subsequent replies, one by Krakowski and another by Swart. This discussion is in no way intended to be a comprehensive review of the topic.

In his articles (Tymoczko (1979) and (1980)), Tymoczko accepts that the mathematical question of the 4CT can be regarded as solved. The proof has been accepted by most mathematicians. However, he argues that the 4CT is the "first mathematical theorem to be known a posteriori and raises again for philosophy the problem of distinguishing mathematics from the natural sciences." He identifies three major characteristics of mathematical proofs. Firstly, proofs are "convincing". He accepts that the proof of the 4CT is convincing to mathematicians. Secondly, proofs are "surveyable", that is "a proof is a construction that can be looked over, reviewed, verified by a

rational agent." "To say that (proofs) can be surveyed is to say that they can be definitively checked by members of the mathematical community", and he claims further that "because of surveyability, mathematical theorems are credited by some philosophers with a kind of certainty unobtainable in other sciences. Mathematical theorems are known a priori." With regard to the 4CT, Tymoczko argues that the proof has not been surveyed "in its entirety". "It has not been checked, step by step, as all other proofs have been checked." Tymoczko's third characteristic of proofs is "formalisability". He claims that "most mathematicians and philosophers believe that any acceptable proof can be formalised." Although he concedes that "most mathematicians would concur that there is a formal proof of the 4CT in an appropriate Graph Theory", he suggests that "we believe that the formal proof exists only because we accept the appeal to computers."

And, with regard to the computer calculations, he suggests that "our knowledge rests on general empirical assumptions about the nature of computers and particular empirical assumptions about Appel and Haken's computer work." Moreover, in view of the inherent complexity of the 4CT, he suggests that "The only route to 4CT that we can ever take appears to lead through computer experiments." He concludes that the 4CT is an "a posteriori truth and not an a priori one."

In his reply to Tymoczko's article, Krakowski (1980) argues that the computer proof of the 4CT "introduces nothing new of philosophical importance" (emphasis in original). He argues against an independent notion of "surveyability". He suggests that Tymoczko's notion of "surveyability" reduces to the existence of a "step by step method

by which mathematicians can check proofs" and that "the computer has, in a step by step fashion, surveyed and proved" the crucial lemma in the proof of the 4CT. While it may be true that the "4CT proof can be surveyed by no mathematician" (emphasis in original), he claims that this "is a matter of empirical accident" and that "there is no principled reason why, when longevity reaches astronomical proportions, a bright young mathematician could not spend a few millenia going through the entire proof." On the question of computer reliability he suggests that a computer is no more fallible than a human mathematician. He concludes that "all mathematical theorems can be known a priori" (emphasis in original) and that "the proof highlights the already existing empirical elements of mathematical knowledge" (emphasis in original).

Swart (1980) also argues that the computer proof does not require te revision of the idea that mathematical theorems are a priori truth. He defines an a priori truth as one whose validity can in principle (though not necessarily in practice) be established "without recourse to sense experience of the physical world." This, he claims, does not mean that mankind need not resort to an experiment to know the truth of an a priori truth. He gives four categories of proof involving the sort of case testing required in the proof of the 4CT.

" (i) Those theorems in which the case testing can be done in our heads.

(ii) Those theorems in which the case testing is impossible to carry out without the help of pencil and paper.

(iii) Those theorems in which the testing can be carried out with immense effort by means of pencil and paper - requiring, say, several thousand man hours of effort.

(iv) Those theorems which are entirely beyond the reach of hand calculations and for which the case testing <u>has</u> to be carried out by computer" (emphasis in original).

"The divisions between these categories are not clear cut" and Swart maintains that from the philosophical point of view the division between category (i) and all the rest is the most significant. In these considerations, the fact that the proof is by case testing is not important. The categories apply equally well with "case testing" replaced by "functional analytic estimates." It is interesting to note that Lanford's proof of the Feigenbaum Conjectures is borderline (iii)-(iv) while our proof is undoubtably in category (iv). Swart defines an a posteriori truth as one "whose truth is contingent upon the nature of the universe to which it applies and cannot in principle be known without carrying out at least some experiments." For him, mathematical theorems remain a priori truths. However, Swart concedes that "perhaps (mathematicians) need a new kind of entity that lies somewhere between a theorem and a conjecture. Perhaps these additional entities could be called agnograms, meaning whereby theoremlike statements that we have verified as best we can but whose truth is not known with the kind of assurance we attach to theorems and about which we must remain, to some extent, agnostic."

I have tried to summarise three views of the philosophical implications of computers proofs and the proof of the 4CT in particular. While there are differences between the combinatorial case testing required for the proof of the 4CT and the numerical-functional-analysis-type proofs we are considering, these

considerations carry over. One point however. Tymoczko lays quite a bit of stress on the fact that, by its very nature, the 4CT probably cannot be proved accept using a computer. While, remaining somewhat agnostic as to whether a "computer-free" proof will be found, I feel that it certain that proofs involving the delicate functional analytic estimates we have obtained, must use computer calculations.

5.2 Are these proofs "good" or "bad" mathematics?

In my opinion, the question of whether theorems proved using computer estimates constitute a priori or a posteriori truth is not the most important matter. For the working mathematician a theorem/proof is judged by other criteria. Elegance, applicability and depth are a few of these.

(i) How elegant is the proof?

Although there is a great deal of cleverness in the construction of the numerical functional analysis routines, I can see little intrinsic merit in a proof that depends so decisively on a large number of "number crunching" calculations. Perhaps this is in part a manifestation of the snobbishness of modern mathematicians who regard a decimal point in a proof with horror (a feeling not shared by Soviet mathematicians). In fact, calculation is a great mathematical tradition. Gauss was a prodigious calculator. Indeed, I regard it as rather fitting that a computer is used in the solution of Feigenbaum-like problems. After all the discovery of the phenomena which the theory explains was only made possible with the advent of digital computers! Moreover, with the horsehoes and strange attractors that abound in modern dynamical systems, we should

indeed be glad there is room for any quantitative analysis at all! Maybe for some, to adapt Poincaré, computers and computer proofs are a "desease from which mathematics will recover," but I suspect that this is false and would be a great loss if it were so.

(ii) How applicable is the proof?

The methods of numerical functional analysis and rigorous interval arithmetic have been used to solve a number of Feigenbaum-like problems. After Lanford's proof of the Feigenbaum Conjectures (Lanford (1982)), Eckmann, Koch and Wittwer (1982) proved universality for period doubling in conservative systems. McKay and Percival (1985) have applied rigorous interval arithmetic to obtain good rigorous upper bounds for the "onset of chaos" in the standard map. However, there must come a time when this type of proof will cease to be either practical or profitable. Indeed, we may well have reached this limit already. We have tried (and failed) to adapt the methods for the renormalisation scheme developed by McKay (1983) for the breakdown of invariant circles in conservative systems. The problem was too large (primarily in storage space) for the resources we had available. Another problem with this type of proof is that it is very specific. For example, although, the method can be applied to solve the renormalisation problem for any periodic rotation number, there does not seem to be a way of dealing with a whole class of rotation numbers at one time. This has meant that only one rotation number has generally been dealt with, almost exclusively, the golden mean. These are serious flaws in the method and although it is in principle possible to adapt the method to other renormalisation schemes I seriously doubt whether it is a useful thing to do.

(iii) How deep is the proof?

By this I mean, how much insight does the proof give to the actual problem? I think it is clear from the quote from Lanford given above, that the proof is seriously defficient in this respect. Apart from the importance of "analyticity improvement" on some domain, there are few clues which will enable us to determine under what conditions Feigenbaum-like functional equations will possess analytic solutions. When a geometric construction for the renormalisation transformation can be found (as in the Feigenbaum and circle map cases), for which a fixed point (or periodic point or whatever) is plausible, then it is certainly a strong indication that it actually exists. At its most basic level, the proof is an application of the contraction mapping theorem on an open set in a Banach space. The sophistication comes in obtaining the contraction map and showing that it is indeed a contraction. The fact that such a contraction exists, however, is really a result of the fact that the renormalisation transformation has a fixed point (and 1 is not in the spectrum of the derivative), rather than the other way round. With the right amount of resources, this method can be used to solve any other such problem provided it has a solution, but gives virtually no information as to why the solution should exist!

6. References

UNIX Manuals (versions 4.1 and 4.2 bsd), Bell Laboratories.

VAX-11 Architecture Reference Manual (Revision 6.1, 20.5.82), DEC Document Number EK-VAXAR-RM-001, Digital Equipment Corporation.

VAX Architecture Guide, Digital Equipment Corporation.

Appel, K.I. and W. Haken, 1976, "Every Planar Map is 4-Colorable," B.A.M.S., vol. 82, pp. 218 - 297.

Appel, K.I., W. Haken, and J. Koch, 1977, "Every Planar Map is 4-Colorable: Part 1: Discharging and Part2: Reducibility," Illin. J. Math., vol. 21, pp. 429 - 567.

Arnold, V.I., 1965, "Small Denominators I; on the mapping of a circle into itself," Trans. A.M.S., vol. 46, pp. 213 - 284.

Arnold, V.I., 1983, Geometric Methods in the Theory of Ordinary Differential Equations, Springer Verlag.

Bohr, T., 1984, "A Bound for the Existence of Invariant Circles in a Class of Two Dimensional Dissipative Maps," Preprint L.A.S.S.P. Cornell University.

Campanino, M. and H. Epstein, 1981, "On the Existence of Feigenbaum's Fixed Point," Comm. Math. Phys., vol. 79, pp. 261 - 302.

Collet, P. and J.-P. Eckmann, 1978, "A Renormalization Group Analysis of the Hierachical Model in Statistical Physics," in Lecture Notes in Physics, vol. 74, Springer Verlag.

Collet, P., J.-P. Eckmann, and O.E. Lanford (III), 1980, "Universal Properties of Maps on an Interval," Comm. Math. Phys., vol. 76, pp. 211 - 254.

Collet, P. and J.-P. Eckmann, 1980, Iterated Maps on the Interval as Dynamical Systems, Birkhaeuser.

Cornfeld, I.F., S.V. Fomin, and Ya.G. Sinai, 1982, Ergodic Theory, Springer Verlag .

Deligne, P., 1977, "Les Diffèomorphismes du Cercle [d'après M. R. Herman]," in L.N.M., vol. 567, pp. 99 - 121.

Dieudonnė, J., 1960, Foundations of Modern Analysis, Academic Press.

Di Prima, R.C. and H.L. Swinney, 1981, "Instabilities and Transition in Flow Between Concentric Rotating Cylinders," in Hy-

drodynamic Instabilities and the Transition to Turbulence, ed. H.L. Swinney and J.P. Gollub, Topics in Applied Physics, vol. 45, Springer Verlag.

Eckmann, J.-P., 1981, "Roads to Turbulence in Dissipative Dynamical Systems," Reviews of Modern Physics, vol. 53 No. 4 Part 1, pp. 643 - 654.

Eckmann, J.-P., H. Koch, and P. Wittwer, 1982, "A Computer Assisted proof of Universality for Area Preserving Maps," University of Geneva Preprint UGVA - DPT 1982/04 345.

Feigenbaum, M.J., 1978, "Quantitative Universality for a Class of Nonlinear Transformations," J. Stat. Phys., vol. 19, pp. 25 - 52.

Feigenbaum, M.J., L.P. Kadanoff, and S.J. Shenker, 1982, "Quasi-Periodicity in Dissipative Systems: A Renormalisation Group Analysis," Physica D, vol. 5D, pp. 370-386.

Fein, A.P., M.S. Heutmaker, and J.P. Gollub, 1985, "Scaling at the Transition from Quasiperiodicity to Chaos in a Hydrodynamic System," Physica Scripta, vol. 79, pp. 79 - 84.

Fisher, M. E., 1983, Scaling, Universality and Renormalisation Group Theory, Lecture Notes in Physics, 186, Springer Verlag.

Guckenheimer, J. and P. Holmes, 1983, Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields, Applied mathematical Sciences, 42, Springer Verlag.

Hardy, G.H. and E.M. Wright, 1954, An Introduction to the Theory of Numbers, Oxford University Press.

Hartmann, P., 1964, Ordinary Differential Equations, John Wiley.

Herman, M.R., 1976, "Sur la conjugaison différentiable des difféomorphismes du cercle à des rotations," Publ. Math. I.H.E.S., vol. 49.

Herman, M.R., 1977, "Mésure de Lesbesgue et Nombre de Rotation," in L.N.M., vol. 597, pp. 271 - 293, Springer Verlag.

Hirsch, M.W., C.C. Pugh, and M. Shub, 1977, Invariant Manifolds, L.N.M., 583, Springer Verlag.

Hogan, T., 1984, The C Programmers Handbook, Brady (Prentice-Hall).

Irwin, M.C., 1980, Smooth Dynamical Systems, Academic Press.

Jensen, M.H., P. Bak, and T. Bohr, 1984, "I Circle Maps and II Josephson Junctions, Charge Density Waves and Standard Maps," Preprints.

Jonker, L. and D.A. Rand, 1983, "Universal Properties of Maps of the Circle with Epsilon-Singularities," Comm. Math. Phys., vol. 90, pp. 273-292.

Kato, T., 1976, Perturbation Theory for Linear Operators, Springer Verlag.

Kernighan, B.W. and D.M. Ritchie, 1978, The C Programming Language, Prentice-Hall.

Khinchin, A., 1964, Continued Fractions, Chicago University Press.

Knuth, D.E., 1969, The Art of Computer Programming: 2 - Seminumerical Algorithms, Addison Wesley.

Krakowski, I., 1980, "The Four Color Problem Reconsidered," Philosophical Studies, vol. 38, pp. 91 - 96.

Krasnoselskii, M.A., G.M. Vainikko, P.P. Zabreiko, Ya.B. Rutitskii, and V.Ya. Stetsenko, 1972, Approximate Solution of Operator Equations, Wolters-Noordhoff.

Landau, L.D. and E.M. Lifshitz, 1959, Fluid Mechanics, Pergamon Press.

Lanford (III), O.E., 1981, "Smooth Tranformations of Intervals," in Seminaire Bourbaki 1980/81 No. 563, L.N.M., vol. 901, pp. 36 - 54, Springer Verlag.

Lanford (III), O.E., 1982, "A Computer Assisted Proof of the Feigenbaum Conjectures," B.A.M.S., vol. 6 No.3, pp. 427-434.

Lanford (III), O.E., 1984, "Functional Equations for Circle Homeomorphisms with Golden Rotation Number," J. Stat. Phys., vol. 34 Nos. 1/2, pp. 57 - 73.

Lanford (III), O.E., 1985, "A Numerical Study of the Likelihood of Phase Locking," Physica D, vol. 14D No. 3, pp. 403 - 408.

Lang, S., 1962, Introduction to Differentiable Manifolds, Interscience(John Wiley).

Manton, N.S. and M. Nauenberg, 1983, "Universal Scaling Behaviour for Iterated Maps in the Complex Plane," Comm. Math. Phys., vol. 89, pp. 555 - 570.

Marsden, J.E. and M. McCracken, 1976, The Hopf Bifurcation and its Applications, Springer Verlag.

McKay, R.S., 1982, Ph.D. Thesis, Princeton University.

McKay, R.S., 1983, "A Renormalisation Approach to Invariant Circles in Area-Preserving Maps," Physica D, vol. 7D, pp. 283 - 300.

McKay, R.S. and I.C. Percival, 1985, "Converse KAM: Theory and Practice," Comm. Math. Phys., vol. 98, pp. 469 - 512.

Mestel, B.D., 1982, "An Exposition of Lanford's Proof of the Feigenbaum Conjectures, with the Computer Estimates based on a Method of Eckmann, Koch and Wittwer," M. Sc. Dissertation - Warwick University.

Mestel, B.D., 1984, "A Computer Assisted Proof of Universality for Cubic Critical Maps of the Circle with Rotation Number the Golden Mean," Preprint Warwick University.

Moore, R.A., 1966, Interval Analysis, Prentice Hall.

Nauenberg, M., 1982, "On Fixed Points for Circle Maps," Phys. Lett. A, vol. B92(7), pp. 317 - 320.

Nitecki, Z., 1971, Differentiable Dynamics, M.I.T. Press.

Ostlund, S., D.A. Rand, J. Sethna, and E. Siggia, 1983, "Universal Properties of the Transition from Quasi-Periodicity to Chaos in Dissipative Systems," Physica D, vol. 8D, pp. 303-342.

Palis, J. and W. De Melo, 1982, Geometric Theory of Dynamical Systems: An Introduction, Springer Verlag.

Rand, D.A., 1983, Universality and Renormalisation in Dynamical Systems: Parts I and II, Summer School on Dynamical Systems, International Centre for Theoretical Physics, Trieste, Italy.

Rand, D.A., 1984, "Universality for Critical Golden Circle Maps and the Breakdown of Dissipative Golden Invariant Tori," Preprint L.A.S.S.P. Cornell University.

Rosenberg, H., 1977, "Les Difféomorphismes du Cercle [d'après M. R. Herman]," in L.N.M., vol. 567, pp. 81 - 98.

Ruelle, D. and F. Takens, 1971, "On the Nature of Turbulence," Comm. Math. Phys., vol. 20, pp. 167 - 192.

Shenker, S.J., 1982, "Behaviour in a map of a circle onto itself: Empirical Results," Physica D, vol. 5D, pp. 405-411.

Smale, S., 1980, The Mathematics of time, Springer Verlag.

Swart, E.R., 1980, "The Philosophical Implications of the Four Color Problem," Am. Math. Mnth., vol. 87 No. 9, pp. 697 - 707.

Swinney, H.L. and J.P. Gollub, 1978, "The Transition to Turbulence," Physics Today, vol. 31 No. 8, p. 41.

Swinney, H.L. and J.P. Gollub, eds., 1981, Hydrodynamic Instabilities and the Transition to Turbulence, Topics in Applied Physics, 45, Springer Verlag.

Swinney, H.L., 1983, "Observations of Order and Chaos in Non-Linear Systems," Physica D, vol. 7D, pp. 3 - 15.

Tymoczko, T., 1979, "The Four-Color Problem and its Philosophical Significance," Journal of Philosophy, vol. LXXVI No. 2.

Tymoczko, T., 1980, "Computer Proofs and Mathematicians: A Philosophical Investigation of the Four-Color Problem," Mathematics Magazine, vol. 53 No. 3.

Wagner-Dobler, F., 1985, C Language, Pitman Computer Handbooks, Pitman.

Whitley, D., 1983, "Discrete Dynamical Systems in Dimensions One and Two," Bull. Lon. Math. Soc., vol. 15, pp. 177 - 217.

Widom, M., 1983, "Renormalization Group Analysis of Quasi-Periodicity in Analytic Maps," Comm. Math. Phys., vol. 92, pp. 121 - 136.

Wilkinson, J.H. and C. Reinsch, 1971, Linear Algebra: A Handbook of Automatic Computation, Grundlehren der Mathematischen Wissenschaften, 186, Springer Verlag.

Yoccoz, J.C., 1984 , "Conjugaison Différentiable des Difféomorphismes du Cercle dont le Nombre de Rotation verifie une condition diophantienne," preprint.

Zeeman, E.C., 1977, Catastrophe Theory: Selected papers 1972 - 1977, Addsion-Wesley.

Zeeman, E.C., 1982, "Bifurcation and Catastrophe Theory," Contemporary Mathematics, vol. 9, pp. 207 - 272.

A0. Notation

This appendix contains some of the notation used commonly in the text.

A	The Banach space $A(\Omega_1) \times A(\Omega_2)$.
A^3	The Banach space of pairs $(\xi, \eta) \in A$, for which both ξ, η are analytic functions of x^3.
A^{com}	The space A with the commuting conditions (3.4).
A^{com}_{cr}	The space A with the conditions (3.4) and (3.5).
$A(\Omega)$	Banach space of real analytic functions on Ω, with the supremum norm $\|.\|_\Omega$.
$A(a, r)$	Banach space of real analytic function on $D(a, r)$ with the supremum norm $\|.\|_{D(a,r)}$.
a_1, r_1	Centre and radius of the domain of h
a_2, r_2	Centre and radius of the domain of k.
β	The scaling parameter for the map T.
β_*	The scaling parameter for the critical fixed point (ξ_*, η_*).
C	The renormalisation transformation with constant $\beta = \beta_*$ given by (A4.4).
$C(a, r)$	$\{ z \in \mathbb{C} : \|z - a\| = r \}$
$cl(A)$	The closure of the set A.
D	Differentiation of a function with respect to its argument.
$D(a, r)$	$\{ z \in \mathbb{C} : \|z - a\| < r \}$
ε_*	The leading eigenvalue of dT_*
$(\delta h, \delta k)$	Tangent vector at $(h, k) \in A(a_1, r_1) \times A(a_2, r_2)$ or $L(a_1, r_1) \times L(a_2, r_2)$.

$(s\xi, s\eta)$	Tangent vector at $(\xi, \eta) \in A$.
dT_*	The derivative of T at the critical fixed point (ξ_*, η_*).
F_n	The nth Fibonnacci number.
$f_{\omega,a}$	The two parameter sine map (2.1).
g	The function $\beta_*^{-1}k_*$.
$<$	For open sets A, B, $A < B$ iff $cl(a) \subseteq B$.
(h, k)	Pair of functions associated with (ξ, η): $\xi(x) = h(x^3)$, $\eta(x) = k(x^3)$.
(h_*, k_*)	The critical fixed point of T: $\xi_*(x) = h_*(x^3)$, $\eta_*(x) = k_*(x^3)$.
κ	The function $\beta_*^{-1}\eta_*$.
ℓ_1	Banach space of real sequences with bounded L^1-norm.
$L(a, r)$	Banach space of real analytic functions on $D(a, r)$ with L^1-norm $\|.\|_{a,r}$.
$\Lambda(E, F)$	The space of bounded linear maps from a Banach space E to F.
N	The approximate Newton map for T.
N_i	The approximate Newton map for S.
Ω_1, Ω_2	The domains of definition of the pair of functions (ξ, η).
p_n/q_n	The rational approximants to a number in $[0, 1)$, usually the golden mean σ.
R_ω	The map $x \to x + \omega$, the lift to R of the rotation through constant angle ω.
$\rho, \rho(f)$	The rotation number.
S	The eigenvalue/vector map (3.4).
σ	The golden mean $\sigma = (\sqrt{5} - 1)/2$.
T	Renormalisation transformation. The letter T is used

to denote each of the transformations (2.18), (3.1) and (A6.7).

W^s, W^u — Stable and unstable manifolds.

(ξ, η) — Pair of functions often thought of as forming a map of the circle.

(ξ_*, η_*) — The critical fixed point of T.

(ξ_s, η_s) — The simple fixed point of T.

A1. Continued Fractions

A1.1 Simple Continued Fractions

This appendix contains a brief review of some of the properties of simple continued fractions that we use. Good references are Khinchin (1964) and Hardy and Wright (1954).

A continued fraction is an expresion of the form:

$$\alpha = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}} \qquad \text{(A1.1)}$$

which is usually written $[a_1, a_2, a_3, \ldots]$. We shall only consider simple continued fractions where the a_i are positive integers. Every $\alpha \in [0,1)$ has a simple continued fraction expansion. If $\alpha \in \mathbb{Q}$ then the continued fraction terminates after a finite number of steps; if α is irrational the expansion is infinite. The expansion is unique if α is irrational and is almost unique if α is rational (in which case case there are two possiblities, for $[a_1, \ldots, a_m, 1] = [a_1, \ldots, a_m+1]$). If we terminate (A1.1) at the nth stage we obtain the a rational number $[a_1, a_2, \ldots, a_n]$ which we write as p_n/q_n, with p_n and q_n relatively prime. Writing $p_0 = 0$, $p_1 = 1$ and $q_0 = 1$, $q_1 = a_1$ then we have

$$\begin{bmatrix} p_{n+1} \\ p_n \end{bmatrix} = \begin{bmatrix} a_n & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_n \\ p_{n-1} \end{bmatrix}, \quad \begin{bmatrix} q_{n+1} \\ q_n \end{bmatrix} = \begin{bmatrix} a_n & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} q_n \\ q_{n-1} \end{bmatrix} \qquad \text{(A1.2)}$$

A1.2 Eventually Periodic Continued Fractions

Two infinite continued fractions, $\alpha = [a_1, a_2, \ldots]$ and $\beta = [b_1, b_2, \ldots]$ have the same tail if there are $n, m > 0$ such that $a_{n+r} = b_{m+r}$ for $r = 1, 2, \ldots$ Two numbers α and β have the same tail if, and only if,

there are integers c, d, e, f such that $(c\alpha + d)/(e\alpha + f) = \beta$ (Hardy and Wright (1954), Theorem 175). A continued fraction $\alpha = [a_1, a_2, \ldots]$ has a <u>periodic tail</u> (or is <u>eventually periodic)</u> if there are $n, r > 0$ such that $a_{m+r} = a_m$ for all $m > n$. In this case r is the period of the tail. In fact, α has a periodic tail if, and only if, it is a quadratic irrational, that is, a root of a quadratic equation with integer coefficients (Khinchin (1964), Theorem 28). We are particularly interested in those α with periodic tails.

Suppose $\alpha = [a_1, a_2, \ldots]$ is eventually periodic with period r. Let n be such that $a_{m+r} = a_m$ for $m > n$. Then equations (A1.2) give

$$\begin{bmatrix} q_{m+rk+1} \\ q_{m+rk} \end{bmatrix} = A^k \begin{bmatrix} q_{m+1} \\ q_{m-1} \end{bmatrix} \qquad \text{(A1.3)}$$

for $k > 1$, where A is the matrix:

$$\prod_{i=n+1}^{n+r} \begin{bmatrix} a_i & 1 \\ 1 & 0 \end{bmatrix}$$

The same equation holds with the q's replaces by p's.

Now, $\det(A) = (-1)^r$, $\mathrm{Tr}(A) > 0$ and A is symmetric so that the eigenvalues of A, λ_1, λ_2 are real and satisfy:

$$0 < |\lambda_1| < 1 < |\lambda_2| \text{ and } \lambda_1 = (-1)^r.\lambda_2^{-1}$$

Writing the right hand side of (A1.3) in terms of the eigenvectors of A, we obtain relations:

$$q_{m+kr} = C_1.\lambda_1^k + C_2.\lambda_2^k \; ; \; p_{m+kr} = D_1.\lambda_1^k + D_2.\lambda_2^k \qquad \text{(A1.4)}$$

for $k > 0$, where the constants C_1, C_2, D_1, D_2 depend only on $m > n$. Using the fact that $p_n/q_n \to \alpha$, we obtain the relation $C_1.\alpha - D_1 = 0$ and the following scaling relations are immediate:

$$\begin{aligned} q_{m+rk}.\alpha - p_{n+rk} &\sim \lambda_1^k && \text{as } k \to \infty \\ p_{m+rk}/q_{m+rk} - \alpha &\sim ((-1)^r\lambda_1^2)^k && \text{as } k \to \infty. \end{aligned} \qquad \text{(A1.5)}$$

A1.3 The Golden Mean σ

Of particular importance is the golden mean $\sigma = (\sqrt{5} - 1)/2$, which has continued fraction exapnsion [1, 1, 1, ...]. (Note: Most people refer to $(\sqrt{5} + 1)/2 = 1 + \sigma$ as the golden mean.) Mckay (1982) has christened numbers that have tail [1, 1, ...] noble numbers. These numbers are important because they are the numbers least well approximable by rational numbers and σ is the least well approximable of all.

σ satifies the equation $\sigma^2 = 1 - \sigma$ and $q_n = p_{n+1} = F_n$, where F_n, $n > 0$ are the Fibonacci numbers 1, 1, 2, 3, 5, 8, 13, ...

In this case the eigenvalues λ_1, λ_2 are $-\sigma$, $1/\sigma$ respectively and the equations (A1.5) become

$$\begin{aligned} q_k.\sigma - p_k &\sim (-\sigma)^k \quad \text{as } k \to \infty \\ p_k/q_k - \sigma &\sim (-\sigma^2)^k \quad \text{as } k \to \infty. \end{aligned} \tag{A1.6}$$

A2. Maps of the Circle

This appendix contains a review of the basic theory of circle maps. Some references to the subject are Herman (1976), Cornfeld et al (1982) and Nitecki (1971).

We denote the circle $\mathbb{R}/\mathbb{Z}$ by T^1. Note that our circles have length 1 rather than 2π. $\mathbb{R}$ is the universal cover of the circle and any map $f: T^1 \to T^1$ lifts to a unique map $f: \mathbb{R} \to \mathbb{R}$ satisfying $f(0) \in [0, 1)$ and $f(x + 1) = f(x) + 1$. It is usually convenient to work with the lift and we shall identify a circle map with its lift and use the same notation throughout. Which one is being referred to at any time will be easily discernable from the context. We shall always assume that f is a homeomorphism of T^1.

One particular example of a circle map is a simple rotation through a constant angle ω, whose lift is $R_\omega: x \to x + \omega$.

Poincaré first introduced the notion of rotation number ρ of a circle map f, defined in terms of the lift by

$$\rho(f) = \lim_{n\to\infty} \frac{(f^n(x) - x)}{n} \qquad \text{(A2.1)}$$

This limit exists and is independent of x. The following are some properties of ρ.

(a) $\rho(f) \in [0, 1)$

(b) $\rho(R_\omega) = \omega$

(c) If h is a homeomorphism of the circle then $\rho(f) = \rho(h^{-1} \circ f \circ h)$.

The dynamics of a circle map depend crucially on the rationality or irrationality of the rotation number ρ. For generic f we have the following situation:

If $\rho(f) = p/q \in \mathbb{Q}$ then every point of T^1 is attracted to a periodic orbit of period q. If $\rho(f) \notin \mathbb{Q}$, then there are two possibilities: Either

every point has a dense orbit in T^1, in which case f is topologically conjugate to the rotation $R_{\rho(f)}$, or f has a Denjoy interval I with the property that $f^n(I) \cap I = \emptyset$, for each $n \neq 0$. In the second case every point is attracted to an invariant Cantor set. Denjoy's Theorem states that if f is a C^2-diffeomorphism with $\rho(f) \notin \mathbb{Q}$ then this second case is not possible, and f is topologically equivalent to a rotation (i.e. there is a homeomorphism h of T^1 satisfying $h^{-1} \circ f \circ h = R_{\rho(f)}$.) Unfortunately, Denjoy's theorem does not apply to maps with a single cubic critical point (which we shall refer to as cubic critical maps). However, it is shown in Ostlund et al (1983) that for a large class of cubic critical maps (those in the stable manifold of the critical fixed point of T) there cannot be a Denjoy interval and so they will be topologically conjugate to a rotation.

For f differentiable, it is an natural question to ask whether or not the map h conjugating f to a rotation is itself differentiable. This depends on the arithmetic properties of $\rho(f)$.

Definition: An irrational α is diophantine if there are constants C, $\epsilon > 0$ such that for all $p, q \in \mathbb{Z}$,

$$|\alpha - p/q| > 1/q^{2+\epsilon} \qquad \text{(A2.2)}$$

We denote by D the set of diophantine numbers. D has measure 1 in [0,1) and contains the quadratic irrationals i.e those numbers whose continued fraction expansion is eventually periodic (see Appendix 1).

Definition: The Liouville numbers are $\mathbb{R} \setminus (\mathbb{Q} \cup D)$.

Arnold has constructed an example of an analytic diffeormorphism f with Liouville rotation number which is C^0 but not C^1 conjugate to a rotation. However Herman showed that for a set of numbers of full measure the homeomorphism h was differentiable. Yoccoz has recently extended this result to include all diophantine numbers.

Theorem A2.1 (Herman-Yoccoz, see Yoccoz (1984)) Let f be a C^∞ (resp. C^ω) diffeomorphism of the circle and let $\rho(f) \in D$. Then f is C^∞ (resp. C^ω) conjugate to $R_{\rho(f)}$.

□

For a given f we consider the family $f_\omega = R_\omega \circ f$. Herman (1977) has shown that if f is a C^1 diffeomorphism which is C^1 conjugate to $R_{\rho(f)}$ then the map $\omega \to \rho(f_\omega)$ is differentiable at $\omega = 0$, with non-zero derivative.

For diffeomorphisms of the circle we may use the Herman-Yoccoz theorem to deduce some scaling laws:

Proposition A2.2 Let f be a C^∞ of the circle with $\rho(f)$ a quadratic irrational. Let r, λ_1 be the values given in Appendix 1. Then (in terms of the lift)

$$f^{qkr}(0) - p_{kr} \sim \lambda_1^k . \qquad (A2.3)$$

Moreover, let ω_k be a sequence for which $\rho(R\omega_k \circ f) = p_k/q_k$. Then

$$\omega_k \sim ((-1)^r\lambda_1^2)^k. \qquad (A2.4)$$

proof Let $\alpha = \rho(f)$. By Herman's theorem, there is a homeomorphism h of T^1 such that

$$h^{-1} \circ f \circ h = R_\alpha .$$

With no loss of generality we may take $h(0) = 0$. Hence

$$h^{-1}(f^{qkr}(0) - p_{kr}) = R_\alpha^{qkr}(0) - p_{kr} = q_{kr}.\alpha - p_{kr}$$

and so $f^{qkr}(0) - p_{kr} \to 0$ as $k \to \infty$. Expanding the left hand side for large k we have

$$Dh^{-1}(0)(f^{qkr}(0) - p_{kr}) = q_{kr}.\alpha - p_{kr} + O((f^{qkr} - p_{kr})^2)$$

and from equation (A1.6) we get

$$f^{qkr}(0) - p_{kr} \sim \lambda_1^k \text{ as } k \to \infty.$$

Now consider the family f_ω as defined above. We know that the function $\omega \to \rho(f_\omega)$ is differentiable at $\omega = 0$. Then as $k \to \infty$, $\omega_k \to 0$ and

$$\rho(f\omega_{kr}) = \alpha + d/d\omega|_{\omega=0}\ \rho(f_\omega).\omega_{kr} + 0(\omega_{kr}^2)$$

Moreover the derivative is non-zero so that

$$\omega_{kr} \sim p_{kr}/q_{kr} - \alpha \sim ((-1)^r\lambda_1^2)^k \quad \text{as } k \to \infty.$$

□

In fact one can prove the following:

Proposition A2.3 Under the hypotheses of Proposition A2.2 we have

$$\lambda_1^{-k}.(f^{q_{kr}}(\lambda_1^k.x) - p_{kr}) \to R_c^r(x) \text{ as } k \to \infty$$

for all $x \in \mathbb{R}$, for some $c \in \mathbb{R}$. (A2.5)

proof Let h be as in the proof of Proposition A2.2. Let $x \in \mathbb{R}$ be fixed. Then

$$\lambda_1^{-k}(f^{q_{kr}}(\lambda_1^k.x) - p_{kr}) = \lambda_1^{-k}(h(h^{-1}(\lambda_1^k.x) + q_{kr}.\alpha - p_{kr})$$

For k large we have

$$\lambda_1^{-k}(Dh(0).D(h^{-1})(0)\ \lambda_1^k.x + Dh(0)(q_{kr}.\alpha - p_{kr})) + 0(\lambda_1^{2k})$$

$$\to x + c \text{ for some constant } c.$$

□

Finally, we give another characterisation of rotation number. For f a homeomorhpism denote by I_0 the interval [0, f(0)] going round the circle. Note that if y is in the interior of I_0, then the next time an iterate of y is in I_0, y has been around the circle precisely once. The rotation number is given by:

$$\rho(f) = \lim_{n\to\infty} m(n)/n \qquad \text{(A2.6)}$$

where $m(n) = \#\{ j : f^j(x) \in I_1,\ j = 1, \ldots, n \}$ and x is a point for which $f^j(x) \neq 0$ for any $j > 0$. That this formula gives the rotation number from the equation follows immediately from

$$m(n) \leq f^n(x) \leq m(n) + 1 .$$

This equation is a result of the fact that m(n) counts the number of times x goes round the circle under n iterations.

A3. Spaces of analytic functions, analyticity improving operators, compact linear operators, spectral perturbation theory and L^1-Spaces

This appendix contains details of the Banach spaces in which we work. This is followed by a discussion of analyticity improving linear operators and their relationship to compact linear operators. Then we review the theory of perturbation of linear operators from Kato (1976) and, finally, we consider L^1-spaces in more detail.

A3.1 Banach Spaces of Analytic Functions

Let Ω be a (simply connected bounded) region in $\mathbb{C}$. Let $\| \cdot \|_\Omega$ denote the supremum norm on Ω given by $\| f \|_\Omega = \sup \{|f(x)| : x \in \Omega\}$. Let $A(\Omega)$ denote the space of real analytic functions $f:\Omega \to \mathbb{C}$ such that f is continuous on $cl(\Omega)$ and hence $\| f \|_\Omega < \infty$. Then $(A(\Omega), \| \cdot \|_\Omega)$ is a real Banach space. A special case occurs when Ω is a disc around a point $a \in \mathbb{C}$. We fix the following notation. Let $r > 0$, $a \in \mathbb{C}$, $D(a, r) = \{z \in \mathbb{C} : |z - a| < r\}$ and $C(a, r) = \partial D(a, r) = \{z \in \mathbb{C} : |z - a| = r\}$, with $C(a, r)$ given the positive orientation. We shall write $A(a, r)$ for the space $A(D(a, r))$.

We find it convenient to work in the L^1-space rather than the "sup" space. Let $f \in A(a, r)$. Then

$$f(x) = \sum_{i=0}^{\infty} f_i (x - a)^i / r^i$$

and the L^1-norm is given by

$$\| f \|_{a,r} = \sum_{i=0}^{\infty} |f_i|$$

Let $L(a, r) = \{f \in A(a, r) : \| f \|_{a,r} < \infty\}$. Then, with this norm, $L(a, r)$ is a real Banach space. We remark that $L(a, r)$ is isomorphic to the space ℓ_1 of sequences $s = (s_0, s_1, \ldots)$ with

$$\| s \| = \sum_{i=0}^{\infty} |s_i| < \infty.$$

The spaces A(a, r) and L(a, r) are related as follows.

Proposition A3.1 Let r, s > 0 and f ∈ A(a, r). Then

(a) $\| f \|_{D(a,r)} \leq \| f \|_{a,r}$ (including the possibility that $\| f \|_{a,r} = \infty$).

(b) $\| f \|_{a,s} \leq K(r, s).\| f \|_{D(a,r)}$ where $K(r, s) = r/(r - s)$.

(c) $A(a, r) \subset L(a, s) \subset A(a, s)$.

proof Let

$$f(x) = \sum_{i=0}^{\infty} f_i(x - a)^i/r^i$$

(a) If $x \in D(a, r)$, then

$$|f(x)| \leq |\sum_{i=0}^{\infty} f_i(x - a)^i/r^i| \leq \sum_{i=0}^{\infty} |f_i| \, |(x - a)/r|^i \leq \| f \|_{a,r}$$

since $|(x - a)/r| < 1$ if $x \in D(a, r)$. Therefore

$$\| f \|_{D(a,r)} \leq \| f \|_{a,r}$$

(b) Using the Cauchy integral formula (see for example Dieudonné (1960)) we have for $i \geq 0$

$$f_i/r^i = 1/2\pi i \int_{C(a,r)} f(z)/(z-a)^{i+1} \, dz$$

so that

$$|f_i| \leq \| f \|_{D(a,r)}$$

Then since

$$f(x) = \sum_{i=0}^{\infty} f_i \, (s/r)^i \, (x - a)^i/s^i$$

we have that

$$\| f \|_{a,s} = \sum_{i=0}^{\infty} |f_i| \, (s/r)^i \leq \| f \|_{D(a,r)} \sum_{i=0}^{\infty} (s/r)^i \leq \| f \|_{D(a,r)} r/(r - s).$$

(c) The inclusions follow immediately from (a) and (b). We shall not prove that the inclusions are strict. This involves constructing an analytic function in the unit disc with finite supremum norm but with infinite L^1-norm. □

We note that $K(r,s) \to \infty$ as $s \to r$ so that A3.1(b) does not give a bound for $\| f \|_{D(a,r)}$ in terms of $\| f \|_{a,r}$.

A3.2 Properties of the spaces A(a, r) and L(a,r)

Let a, r > 0 be fixed. Let $\| \cdot \|$ denote either the norm $\| \cdot \|_{D(a,r)}$ or $\| \cdot \|_{a,r}$ and let E denote either A(a, r) or L(a, r). Let f, g $\in$ E.

Proposition A3.2 The following hold

(a) $\|f + g\| \leq \| f \| + \| g \|$.

(b) $\|\lambda f\| = |\lambda| . \| f \|$, $\lambda \in \mathbb{R}$.

(c) $\|f.g\| \leq \| f \| . \| g \|$.

(d) If $\|g - a\| < r$, then $\|f \circ g\| \leq \| f \|$.

(e) Let $0 < s < r$ and let $g(D(a,r) \subset D(a,s)$. Then for $r > 0$,

$$\|(D^r f) \circ g\| \leq \| f \| . \sup\{(i+r)\ldots(i+1)s^i : i > 0\}.$$

The proofs of the statements are elementary. Indeed (a), (b) are implicit the statement the $\| \cdot \|$ is a norm.

We remark that the maps

$$S_1 : \mathbb{R} \times E \to E, \ (\lambda, f) \to \lambda f \tag{A3.1}$$

$$S_2 : E \times E \to E, \ (f, g) \to f + g \tag{A3.2}$$

$$S_3 : E \times E \to E, \ (f, g) \to f.g \tag{A3.3}$$

are all C^∞ maps with derivatives

$$dS_1(\lambda,f) : \mathbb{R} \times E \to E \ (\delta\lambda,\delta f) \to \delta\lambda.f + \lambda.\delta f \tag{A3.4}$$

$$dS_2(f,g) : E \times E \to E \ (\delta f,\delta g) \to \delta f + \delta g \tag{A3.5}$$

$$dS_3(f,g) : E \times E \to E \ (\delta f,\delta g) \to \delta f.g + f.\delta g \tag{A3.6}$$

respectively. These are all standard results (see for example Irwin (1980), Appendix B) Also standard is the following

Proposition A3.3 Let $f_0, g_0 \in E$ with $cl(g_0(D(a,r))) \subseteq D(a,r)$. Let $r > 0$. Then the map

$$S_4 : V \to E, \ (f, g) \to (D^r f) \circ g \tag{A3.7}$$

is defined on a neighbourhood of $(f_0, g0)$ in E x E and is C^∞ there.

Its derivative is given by:

$$d\dot{S}_4(f,g) : E \times E \dashrightarrow E$$

$$(\delta f, \delta g) \dashrightarrow D^r \delta f \circ g + (D^{r+1} f \circ g).\delta g \qquad \text{(A3.8)}$$

□

We shall not give a proof of this proposition. A discussion of this type of result is contained in Appendix B of Irwin (1980). Irwin in fact considers more general spaces of C^r functions but the ideas are the same. We remark that it is not necessary that f and g be defined on the same discs. The important property is that g_0 maps the domain of g strictly into the domain of f.

A3.3 Analyticity Improving Maps

A most important property of the renormalisation transformation is that, on properly chosen domains and close to the fixed point, it is an analyticity improving transformation i.e. the transformed functions are analytic on larger domains. This is also true of the derivative and, for linear maps, we formalise this idea as follows. Let Ω be as defined in Section A3.1.

Definition A linear map $B: A(\Omega) \dashrightarrow A(\Omega)$ is analyticity improving if there is another open region Ω' with $cl(\Omega) \subseteq \Omega'$, such that $B(A(\Omega)) \subseteq A(\Omega')$ and $B:A(\Omega) \dashrightarrow A(\Omega')$ is a bounded linear map.

We recall that a linear map B is compact if the image of a bounded set is relatively compact. An equivalent definition is that for any bounded sequence $\{f_n\}$, $\{B(f_n)\}$ has a Cauchy subsequence (see for example Kato (1976)).

Proposition A3.4 (see Jonker and Rand (1983)) Let $B:A(\Omega) \dashrightarrow A(\Omega)$ be an analyticity improving linear map. Then B is compact.

proof Let Ω' be as in the above definition. Let $\{f_n\}$ be a bounded sequence in $A(\Omega)$. With no loss of generality, we may assume that

$\|f_n\|_\Omega < 1$ for each n. Now let C be a closed contour in $\Omega' \setminus cl(\Omega)$ and let x, y $\in \Omega$. Then by the Cauchy integral formula

$$|B(f_n)(x) - B(f_n)(y)| = 1/2\pi i \int_C B(f_n)(z)/(z-x)\, dz - 1/2\pi i \int_C B(f_n)(z)/(z-y)\, dz$$

$$= 1/2\pi i \int_C (x-y)B(f_n)(z) / ((z-x).(z-y))dz$$

$$< 1/2\pi \|B(f_n)\|_{\Omega'} L(C).|x-y|/\varepsilon^2$$

where L(C) in the length of the curve C and $\varepsilon = d(C, \Omega)$, the distance of Ω from the contour C. However B: $A(\Omega) \to A(\Omega')$ is a bounded linear map so that

$$|B(f_n)(x) - B(f_n)(y)| < K(x-y)$$

where K is a constant that <u>independent of n.</u> Thus the family is an equicontinuous family of maps on Ω. Now from Ascoli's theorem (Dieudoné (1960)) we conclude that the sequence $\{B(f_n)\}$ has a Cauchy subsequence in $A(\Omega)$.

□

We shall actually work in the L^1-spaces. We extend the above result to these spaces.

<u>Proposition A3.5</u> Let B: A(a, r) --> A(a, r) be an analyticity improving map. Then $B(L(a, r)) \subseteq L(a, r)$ and the restriction of B to L(a, r) (also written B) is also a compact operator.

<u>proof</u> Let Ω = D(a,r) and let Ω' be as in the definition of analyticity improving map. We take s > r such that $cl(D(a,s)) \subseteq \Omega'$. Now $B(L(a, r)) \subseteq B(A(a, r)) \subseteq A(\Omega') \subseteq A(a, s) \subseteq L(a, r)$ so that $B(L(a, r)) \subseteq L(a, r)$. To show that B: L(a, r) --> L(a, r) is a compact operator, let $\{f_n\}$ be a bounded sequence in L(a, r). Then Proposition A3.1 (a) shows that $\{f_n\}$ is also a bounded sequence in A(a, r). The argument in the proof of Proposition A3.4 now applies for x, y $\in$ D(a, s) to give that $\{B(f_n)\}$ is a Cauchy sequence in A(a, s). Then

A3.1 (b) gives that $\{B(f_n)\}$ is a Cauchy sequence in L(a, r). Hence B: L(a, r) --> L(a, r) is a compact operator.

□

The analyticity improvement linear maps that we deal with are all of the following type:

Proposition A3.6

(a) Let $t > r$, $g_1, g_2 \in A(a, t)$ be such that $\| g_1 - a \|_{D(a,t)}$, $\| g_2 - a \|_{D(a,t)} < r$. Let $h \in A(a, r)$. Then the map B: A(a, r) --> A(a, r) defined by $B(f) = (h \circ g_1).(f \circ g_2)$ is an analyticity improving map B:A(a, r) --> A(a, r) and hence is compact.

(b) Let $t > r$, $g_1, g_2 \in L(a, t)$ be such that $\| g_1 - a \|_{a,t}$, $\| g_2 - a \|_{a,t} < r$. Let $h \in L(a, r)$. Then the map B: L(a, r) --> L(a, r) defined by $B(f) = (h \circ g_1).(f \circ g_2)$ extends to an analyticity improving map B:A(a, r) --> A(a, r) and hence B:L(a, r) --> L(a, r) is compact.

proof

(a) For $f \in A(a, r)$, $B(f) \in A(a, r)$ since

$$\| B(f) \|_{D(a,r)} \leq \| h \circ g_1 \|_{D(a,r)} \cdot \| f \circ g_2 \|_{D(a,r)}$$

$$\leq \| h \|_{D(a,r)} \cdot \| f \|_{D(a,r)} \cdot$$

The condition on the norms of g_1, g_2 is sufficient to guarentee that these compositions are well defined. Now, we have that $\| B(f) \|_{D(a,t)} \leq \| h \|_{D(a,r)} \cdot \| f \|_{D(a,r)}$ so that B is a bounded map from A(a, r) --> A(a, t), that is, B is an analyticity improving map A(a, r) --> A(a, r). It follows that B is compact by Proposition A3.4.

(b) For $f \in L(a, r)$, $B(f) \in L(a, r)$ since

$$\| B(f) \|_{a,r} \leq \| h \circ g_1 \|_{a,r} \cdot \| f \circ g_2 \|_{a,r} \leq \| h \|_{a,r} \cdot \| f \|_{a,r} \cdot$$

Defining B for $f \in A(a, r)$ by $B(f) = (h \circ g_1).(f \circ g_2)$, we see that the condition on the norms of g_1, g_2 is sufficient to guarentee that these compositions are well defined (by Proposition A3.1(a)). Now, we have

that $\| B(f) \|_{D(a,t)} < \| h \|_{D(a,r)} \cdot \| f \|_{D(a,r)}$ so that B is a bounded map from A(a, r) --> A(a, t), that is, B is an analyticity improving map A(a, r) --> A(a, r). It follows that B restricted to L(a, r) is compact by Proposition A3.5.

□

It is easy to see that a linear combination of two analyticity improving maps A(a, r) --> A(a, r) is also analyticity improving.

We work mostly with pairs of maps and the appropriate space to work is the direct sum of these Banach spaces. We denote by $L = L(a_1, r_1) \oplus L(a_2, r_2)$, which is a real Banach space when given the norm $\| (f, g) \| = \| f \| + \| g \|$. A linear operator B: L --> L is compact if, and only if, the linear maps B_{11}, B_{12}, B_{21}. B_{22} are compact where:

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} : L \longrightarrow L$$

A3.4 Spectral Theory and Perturbation Theory

The main reference for this section is the comprehensive account of the perturbation theory for linear maps given in Kato (1976).

Let E be a real Banach space and let B: E --> E be a compact linear operator. We make a number of remarks about the complexification of a real Banach space. The right place to do spectral theory is in a linear space over the complex field. We work in linear spaces over the real numbers. We recall that we can complexify a real space by writing $E^C = E + i.E$ with norm $\|u + i.v\| = \sqrt{(\|u\|^2 + \|v\|^2)}$ for $u, v \in E$. A linear operator B on E induces a linear operator on E^C (which we also write B) by

$$B(u + i.v) = B(u) + i.B(v)$$

As usual, the complex conjugate of u + i.v is defined to be u - i.v. As with finite dimensional spaces, spectral values of a real Banach space are either real or come in complex conjugate pairs. We note that the property of compactness of B carries over to the operator B on E^C. In what follows, we shall assume that E has been complexified as outlined here, and we shall drop the superscript C.

We recall that $\lambda \in \mathbb{C}$ is in the spectrum of a bounded linear operator B if $(B - \lambda I)$ is not an invertible operator.

Proposition A3.7 (see Theorem 11.4.1 of Dieudonné (1960)) Let B be a compact operator on a complex Banach space E. Then

(a) The spectrum S of B is an at most denumerable compact subset of $\mathbb{C}$, each point of which, with the possible exception of 0, is isolated. If E is infinite dimensional, then $0 \in S$.

(b) Each $\lambda \neq 0$ in S is an eigenvalue of B. The spectral subspace N_λ belonging to any eigenvalue λ is closed, finite dimensional and each $u \in N_\lambda$ is a generalised eigenvector i.e. there is $r \geq 1$ such that $(B - \lambda I)^r u = 0$.

Definition Let B be a bounded linear operator on a Banach space E. The resolvent set P(B) is the set of $\zeta \in \mathbb{C}$ for which $(B - \zeta I)^{-1}$ is a bounded linear operator on E. The operator valued function

$$R(\zeta) = (B - \zeta I)^{-1}$$

defined on the resolvent set is called the resolvent of B. The resolvent is an analytic function of ζ on P(B) (see for example Dieudonné (1960) for a discusion of analytic functions between Banach spaces).

Let $\Gamma_1, \Gamma_2, \ldots, \Gamma_s$ be disjoint simple closed contours lying in P(B) and

for each r = 1, ..., s let

$$P_r = -1/2\pi i \int_{\Gamma r} R(\varsigma)\, d\varsigma.$$

Then E may be written as a direct sum

$$E_1 \oplus E_1 \oplus .... \oplus E_s \oplus E_0 \qquad \text{(A3.9)}$$

where for each r = 1, ..., s, $E_r = P_r E$, $B(E_r) \subseteq E_r$, $B(E_0) \subseteq E_0$ and the spectrum of $B|E_r$ consists of the spectrum of B lying inside Γ_r and that of $B|E_0$ consists of the spectrum of B lying outside all of the E_r, r > 1 (for details of this discussion see Kato (1976) III-§6.4).

Proposition A3.8 Let B_t, $t \in [0, 1]$ be a continuous (in the sense of the operator topology) one parameter family of bounded linear operators on E, and let Γ_1, ..., Γ_s be fixed disjoint simple closed contours and suppose that $S(B_t) \cap \Gamma_r = \emptyset$ for all r = 1, ..., s, and $t \in [0, 1]$. Then for all t the spectral decompositions (A3.9) are isomorphic. This says that if t_1, $t_2 \in [0, 1]$ and if T_{t1} has decomposition

$$E_1(t_1) \oplus E_2(t_2) \oplus \ldots \oplus E_0(t_1)$$

and T_{t2} has decomposition

$$E_1(t_2) \oplus E_2(t_2) \oplus \ldots \oplus E_0(t_2)$$

then

$$E_r(t_1) \cong E_r(t_2) \text{ for } r = 0, .., s.$$

In particular if B_0 has a only one simple eigenvalue inside each of Γ_1, ..., Γ_{s-1} with the rest of its spectrum lying inside Γ_s, then the same is true for B_1 and, in view of the fact that we are working in a real Banach Space, the eigenvalues are all real.

This is a consequence of Kato (1976) I-§4.6, III-§6.4, IV-§3.4.

□

A3.5 L^1-spaces

The L^1-spaces described in section A3.1 are all isomorphic to the Banach space ℓ_1, the space of sequences $s = (s_0, s_1, \ldots)$ with

$$\| s \| = \sum_{i=0}^{\infty} |s_i| < \infty.$$

The ismomorphism comes through the selection of the basis $\{((x - a)/r)^i\}$ for $i = 0, \ldots, \infty$ for the space $L(a_1, r_1)$. The isomoprhism is also an isometry i.e. it is norm preserving. The space ℓ_1 has the property of being isometrically isomorphic to a direct sum of itself and either $\mathbb{R}^m$ or ℓ_1 i.e.

$$\ell_1 \cong \ell_1 \oplus \ell_1 \cong \mathbb{R}^m \oplus \ell_1 \text{ for } m > 0$$

(where we give $\mathbb{R}^m$ the L^1 norm). The isomorphisms are simply a matter of reassigning the basis elements of ℓ_1. This is extremely useful. For example, we may consider the space of pairs of maps $L(a_1, r_1) \oplus L(a_2, r_2)$ (with norm $\| \cdot \|_{a_1,r_1} + \| \cdot \|_{a_2,r_2}$) as simply ℓ_1. In fact, when calculating the derivative of the renormalisation transformation we regard the space of pairs as $\mathbb{R}^{81} \oplus \ell_1 \oplus \mathbb{R}^{81} \oplus \ell_1$. However, when we have transformed the derivative matrix to the "p-basis," and "contracted down to a 4 × 4 matrix" (see Appendix 7) we regard this space as $\mathbb{R}^3 \oplus \ell_1$. This simple conceptual translation is a powerful reason for using these L^1-spaces. We wish to discuss the question of changing the basis of ℓ_1 and the way that error is transmitted through such a basis change. We shall use the convention that subscripts and summation range from 0 to ∞ unless specified othewise. Now, let $\{e_i\}$ be a basis for a real Banach space E. Then we may define the L^1 norm <u>with respect to this basis</u> by the formula for $x \in E$

$$\| x \|_e = \Sigma |x_i|, \quad x = \Sigma e_i . x_i$$

Now suppose E is actually equal to the set of $x \in E$, for which $\| x \|_e < \infty$ then the Banach space $(E, \| \cdot \|_e)$ is isometrically isomorphic to

ℓ_1 via isomorphism $(x_0, x_1, \ldots) \to \Sigma\, e_i.x_i$. We shall assume that this is the case for the rest of this section.

Now let B be a linear operator on E. Then, with respect to the basis $\{e_i\}$, B has a matrix representation (B_{ij}) defined by

$$B(e_j) = \Sigma\, e_i.B_{ij}$$

Then for $x = \Sigma e_i.x_i$

$$\| B(x) \|_e = \| \Sigma\, B(e_j).x_j \| = \| \Sigma_j\, (\Sigma_i\, e_i.B_{ij}).x_j \|$$

$$= \Sigma_j\, |\, (\, \Sigma_i\, B_{ij}\, x_j\,)\, |$$

$$\leq \Sigma_j\, \Sigma_i\, |\, B_{ij}\, |.\, |\, x_j\, |$$

$$\leq \sup\, \{\, \Sigma_i\, |\, B_{ij}\, |\, :\, 0 \leq j\, \}\, \Sigma_j\, |\, x_j\, |$$

$$= \sup\, \{\, \Sigma_i\, |\, B_{ij}\, |\, :\, 0 \leq j\, \}\, \| x_j \|_e$$

In fact by considering in turn $x = e_i$, it is easy to show that this is a best possible bound and

$$\| B \|_e = \| B_{ij} \| \sup\, \{\, \Sigma_i\, |\, B_{ij}\, |\, :\, 0 \leq j\, \}.$$

We shall refer to this norm $\| \cdot \|$ on matrices as the "maximum column sum" norm. The fact that this norm is easily calculated is another reason for using the L^1-spaces.

Now let P be a an invertible m × m matrix. Let $\{e_i'\}$ be the basis of E given by

$$e_j' = \Sigma_i\, e_i.P_{ij},\ 0 \leq j < m,\quad e_j' = e_j,\ j \geq m$$

where the summation ranges from i = 0 to m - 1. Let $\| \cdot \|_e'$ denote the L^1 norm with respect to the basis $\{e_i'\}$. Then

$$\| \cdot \|_e.\| P^{-1} \| \leq \| \cdot \|_e' \leq \| \cdot \|_e.\| P \|$$

where the norms $\| P \|$, $\| P^{-1} \|$ refer to the maximum column sum norms of the matrices P, P^{-1}. The norms of $\| \cdot \|_e$ and $\| \cdot \|_e'$ are thus equivalent and the norm $\| \cdot \|_e'$ generates the same topology on the space E as the norm $\| \cdot \|_e$. Note that the norms of P, P^{-1} provide bounds on the transmission of error in the course of the change of basis $\{e_i\} \to \{e_i'\}$. For, if all that is known about a vector

x is a bound on $\| x \|_e$, then, with respect to the basis $\{e_i'\}$, all that can be said about $\| x \|_e'$ is that it is bounded by $\| x \|_e . \| P \|$. Of course, when more specific information is known then it is possible to give better bounds. This is done when changing from the standard basis to the p-basis since we often know that some of the coefficients x_i are 0.

A4. Normalisation Conditions and the Spectrum of dT_*

A4.1. Change in normalisation condition

Let (ξ_*, η_*) be a fixed point for the Renormalisation Transformation T given in equation (2.18). Let β_* be the value of β at (ξ_*, η_*). Let $\lambda \in \mathbb{R}$, $\lambda \neq 0$. Then the pair of maps (ξ_*', η_*') defined on the sets

$$\Omega_1' = \lambda^{-1}.\Omega_1, \quad \Omega_2' = \lambda^{-1}.\Omega_2$$

respectively and given by

$$\xi_*'(x) = \lambda^{-1}.\xi_*(\lambda x), \qquad \eta_*'(x) = \lambda^{-1}.\eta_*(\lambda x)$$

satisfy:

$$\xi_*'(x) = \beta_*^{-1}.\eta_*'(\beta_* x), \quad \eta_*'(x) = \beta_*^{-1}.\eta_*'(\xi_*'(\beta_* x))$$

where $\beta_* = \beta(\xi_*, \eta_*)$.

Of course, the pair (ξ_*', η_*') does not satisfy the normalisation condition (2.13)(i). We now see why it is necessary to impose this normalisation condition. Without it, the transformation T has a whole line of fixed points and consequently has an eigenvalue one in its spectrum. However, although the condition (2.13)(i) is the most natural condition from the point of view of embedding circle maps (as in equation (2.14)), from the computational point of view it is simpler to impose the condition

$$\xi(0) = 1. \tag{A4.1}$$

This is the normalisation condition used by Feigenbaum et al (1982) and McKay (1982). We shall show that this change in normalisation does not make any difference to spectrum of the derivative of the renormalisation transformation. We introduce some notation. For open sets A, B in $\mathbb{C}$, we shall write $A \Subset B$ if $A \subseteq B$ and we write $A \Subset B$ if $A \subseteq B$ and $cl(A) \subseteq B$. A similar convention will hold for $\Supset$ and $\Supset$.

Let $(\xi_*, \eta_*)$and β_* satisfy the equations:

$$\xi_*(x) = \beta_*^{-1}\eta_*(\beta_* x), \quad \eta_*(x) = \beta_*^{-1}\eta_*(\xi_*(\beta_* x)) \tag{A4.2}$$

on the domains Ω_1, Ω_2 respectively and suppose

$$\beta_*.\Omega_1 < \Omega_2, \ \beta_*.\Omega_2 < \Omega_1, \quad \xi_*(\beta_*\Omega_2) < \Omega_2 \tag{A4.3}$$

We denote by C the map

$$C: (\xi, \eta)(x) \dashrightarrow (\beta_*^{-1}.\eta(\beta_* x), \beta_*^{-1}.\eta(\xi(\beta_* x)) \tag{A4.4}$$

We shall show below that C is defined on an open neighbourhood of (ξ_*, η_*) in $A(\Omega_1) \times A(\Omega_2)$. We shall work with the following set of normalisation conditions. Let $a, b \in \mathbb{R}$, a and b not both 0. We consider the normalisation condition

$$a.\xi(0) + b.\eta(0) = 1 \tag{A4.5}$$

We note that the condition (2.13) (i) has $a = 1$, $b = -1$, while (A4.1) has $a = 1$, $b = 0$. We write $P_{a,b}$ for the map

$$P_{a,b}(\xi, \eta)(x) = \lambda^{-1}.(\xi, \eta)(\lambda x) \tag{A4.6a}$$

where

$$\lambda = a.\xi(0) + b.\eta(0). \tag{A4.6b}$$

Note that $P_{a,b}(\xi, \eta)$ satisfies (A4.5) and that $P_{a,b} \circ P_{a,b} = P_{a,b}$ since if (ξ, η) satisfies (A4.5) then $\lambda = 1$. This means that $P_{a,b}$ is a (non-linear) projection. We write

$$T_{a,b} : (\xi, \eta)(x) \dashrightarrow (\beta^{-1}.\eta(\beta x), \beta^{-1}.\eta(\xi(\beta x))) \tag{A4.7a}$$

where

$$\beta = a.\eta(0) + b.\eta(\xi(0)). \tag{A4.7b}$$

Then, at least formally, we have

$$T_{a,b} = P_{a,b} \circ C, \ T_{a,b} \circ P_{a,b} = P_{a,b} \circ C \tag{A4.8}$$

$$P_{a,b} \circ T_{a,b} = T_{a,b} = T_{a,b} \circ P_{a,b}$$

We shall show that the spectrum of $dT_{a,b}$ and dC are identical at (ξ_*, η_*) except that dC has an extra eigenvalue 1.

For convenience we make the conventions $P_{0,0}(\xi, \eta) = (\xi, \eta)$ and $T_{0,0} = C$. There is a technical difficulty about the domain of definition of $P_{a,b}(\xi, \eta)$ which we shall discuss now. The problem is

that the natural domains on which the pair $P_{a,b}(\xi, \eta)$ is defined are $\lambda^{-1}\Omega_1$, $\lambda^{-1}\Omega_2$. However these depend on (ξ, η) and we need to work with fixed domains. We overcome this in the following manner.

Proposition A4.1. Let $a, b \in \mathbb{R}$ and let (ξ_*, η_*) be a fixed point of $T_{a,b}$ defined on two domains Ω_1, Ω_2 in $\mathbb{C}$ and suppose that

$$\beta_*.\Omega_1 \subset \Omega_2, \ \beta_*.\Omega_2 \subset \Omega_1, \text{ and } \xi_*(\beta_*\Omega_2) \subset \Omega_2. \tag{A4.9}$$

Then there are domains Ω_1^+, Ω_1^-, Ω_2^+, Ω_2^- with

$$\Omega_1^- \subset \Omega_1 \subset \Omega_1^+ \quad \text{and} \quad \Omega_2^- \subset \Omega_2 \subset \Omega_2^+$$

so that (ξ_*, η_*) can be extended to Ω_1^+, Ω_2^+. Furthermore, there is a neighbourhood V of (ξ_*, η_*) in $A(\Omega_1^-) \times A(\Omega_2^-)$ such that for all $\Omega_1', \Omega_2' \subseteq \mathbb{C}$ satisfying

$$\Omega_1^- \Subset \Omega_1' \Subset \Omega_1^+ \quad \text{and} \quad \Omega_2^- \Subset \Omega_2' \Subset \Omega_2^+ \tag{A4.10}$$

we have

(i) $T_{a,b} : V \cap (A(\Omega_1') \times A(\Omega_2')) \longrightarrow A(\Omega_1') \times A(\Omega_2')$ is well defined and C^∞.

(ii) $dT_{a,b}(\xi_*,\eta_*)$: $A(\Omega_1') \times A(\Omega_2') \longrightarrow A(\Omega_1') \times A(\Omega_2')$ is a compact linear operator.

(iii) The (generalised) eigenvectors corresponding to non-zero eigenvalues are the same for all the choices of Ω_1', Ω_2'.

proof

From (A4.9) we see that it is possible to find domains Ω_1^+, Ω_1^-, close to Ω_1, and Ω_2^+, Ω_2^-, close to Ω_2, with the properties

$$\Omega_1^- \subset \Omega_1 \subset \Omega_1^+, \quad \Omega_2^- \subset \Omega_2 \subset \Omega_2^+$$

and

$$\beta.\Omega_1^+ \subset \Omega_2^-, \ \beta.\Omega_2^+ \subset \Omega_1^-, \ \xi(\beta.\Omega_2^+) \subset \Omega_2^- \tag{A4.11}$$

where $\beta = \beta_*$, $\xi = \xi_*$. In particular

$$\beta_*.\Omega_1^+ \subset \Omega_2, \ \beta*.\Omega_2^+ \subset \Omega_1, \ \xi_*(\beta_*.\Omega_2^+) \subset \Omega_2$$

so that in view of equations (A4.2) we may extend (ξ_*, η_*) to Ω_1^+, Ω_2^+. Now let Ω_1', Ω_2' satisfy (A4.10). If $(\xi, \eta) \in A(\Omega_1^-) \times A(\Omega_2^-)$ is close to

(ξ_*, η_*), then β, given by equation (A4.7), is close to β_*, so that there is a neighbourhood V of (ξ_*, η_*) in $A(\Omega_{\bar{1}}) \times A(\Omega_{\bar{2}})$ with the property that if (ξ, η) is in $V \cap (A(\Omega_1') \times A(\Omega_2'))$, then (A4.11) holds with $\beta = \beta(\xi, \eta)$ given by (A4.7). In particular

$$\beta.\Omega_1' < \Omega_2', \ \beta.\Omega_2' < \Omega_1', \ \xi(\beta.\Omega_2') < \Omega_2'$$

so that $T_{a,b}$ is well defined an C^∞ on $V \cap (A(\Omega_1') \times A(\Omega_2'))$ and $dT_{a,b}$ is a compact operator on $A(\Omega_1') \times A(\Omega_2')$ (see A3.7). This proves (i) and (ii). To show (iii) we note that if $(\xi, \eta) \in V \cap (A(\Omega_1') \times A(\Omega_2'))$ then a (generalised) eigenvector $(\delta\xi, \delta\eta)$ of $dT_{a,b}(\xi, \eta)$ with eigenvalue λ in $A(\Omega_1') \times A(\Omega_2')$ is also in $A(\Omega_{\bar{1}}^+) \times A(\Omega_{\bar{2}}^+)$. For if $(dT_{a,b}(\xi, \eta) - \lambda)^r(\delta\xi, \delta\eta) = 0$, then

$$(\delta\xi, \delta\eta) = -1/\lambda \sum_{i=0}^{\infty} C_i^r (-\lambda)^{r-i} dT_{a,b}(\xi, \eta)^i (\delta\xi, \delta\eta)$$

which is in $A(\Omega_1') \times A(\Omega_2')$ in view of (A4.11) and (A6.2). Here

$$\delta\beta = a.\delta\eta(0) + b(\delta\eta(\xi(0)) + D\eta(\xi(0)).\delta\xi(0)).$$

□

With this proposition we may prove the following slight generalisation of Lemma 4.2 in Ostlund et al (1983).

Proposition A4.2 (Ostlund et al (1983)) Let (ξ_*, η_*) be a fixed point of $T_{a,b}$ (a, b not both 0) and of C defined on Ω_1, Ω_2. Then the eigenvalues $\lambda \neq 1$ of $dT_* = dT_{a,b}(\xi_*, \eta_*)$ and $dC_* = dC(\xi_*, \eta_*)$ are identical together with their multiplicities. dC_* has an eigenvector with eigenvalue 1 which is an eigenvector with eigenvalue 0 for dT_*. The spectral subspace corresponding to eigenvalue 1 of dT_* has one less dimension than that of dC_*.

proof

Let $\Omega_{\bar{1}}^+, \Omega_{\bar{1}}, \Omega_{\bar{2}}^+, \Omega_{\bar{2}}, V$ be such that (i), (ii) and (iii) of Proposition A4.1 hold for both $C = T_{0,0}$ and $T_{a,b}$. Then, by reducing V if necessary, we may assume that V is a neighbourhood of (ξ_*, η_*) in $A(\Omega_1) \times A(\Omega_2)$ for which λ (given by (A4.6)) is close to 1 (note that

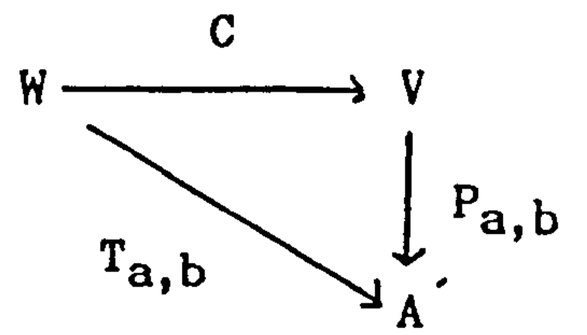
$a\xi_*(0) + b\eta_*(0) = 1$) and such that there are $\Omega_1' \subseteq \{\lambda(\xi, \eta)^{-1}.\Omega_1 : (\xi, \eta) \in V\}$, $\Omega_2' \subseteq \{\lambda(\xi, \eta)^{-1}.\Omega_2 : (\xi, \eta) \in V\}$ that satisfy equation (A4.10). By further reducing V, we obtain a neighbourhood W in $A(\bar{\Omega_1}) \times A(\bar{\Omega_2})$ of (ξ_*, η_*) for which $T_{a,b}(W \cap (A(\Omega_1) \times A(\Omega_2)) \subseteq V \cap (A(\Omega_1) \times A(\Omega_2))$, $C(W \cap (A(\Omega_1) \times A(\Omega_2)) \subseteq V \cap (A(\Omega_1) \times A(\Omega_2))$ and $P_{a,b}(W \cap (A(\Omega_1) \times A(\Omega_2)) \subseteq V \cap (A(\Omega_1') \times A(\Omega_2'))$. For brevity, we shall write

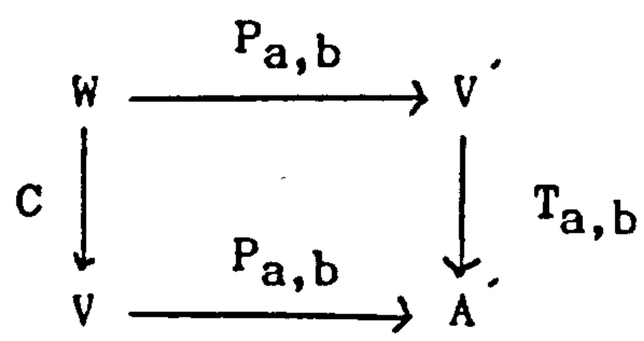
$A = A(\Omega_1) \times A(\Omega_2)$, $A' = A(\Omega_1') \times A(\Omega_2')$

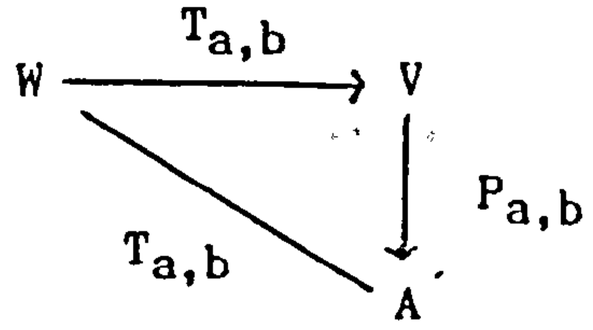
V for $V \cap (A(\Omega_1) \times A(\Omega_2))$, W for $W \cap (A(\Omega_1) \times A(\Omega_2))$

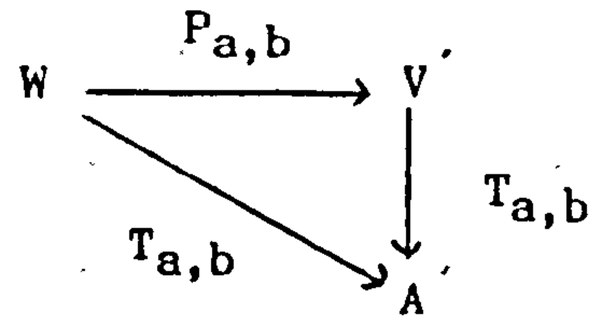
V' for $V \cap (A(\Omega_1') \times A(\Omega_2')$

Then the following diagrams commute:



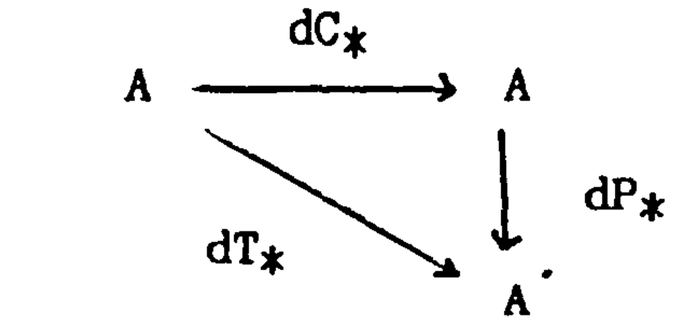






where all of these maps are well defined and C^∞ on their domains.

Differentiating at (ξ_*, η_*) and using the relations $C(\xi_*, \eta_*) = T_{a,b}(\xi_*, \eta_*) = P_{a,b}(\xi_*, \eta_*) = (\xi_*, \eta_*)$ we obtain the following commutative diagrams:



(A4.12)

where $dP_* = dP_{a,b}(\xi_*, \eta_*)$.

Now if $(\delta\xi, \delta\eta)$ is a tangent vector in A, then

$$dP_*(\delta\xi, \delta\eta)(x) = (\delta\xi(x) + (D\xi_*.x - \xi_*(x))\delta\lambda, \qquad (A4.13)$$
$$\delta\eta(x) + (D\eta_*.x - \eta_*(x))\delta\lambda)$$

where $\delta\lambda = a\delta\xi(0) + b\delta\eta(0)$. Now, the right hand side of equation (A4.13) in fact lies in A (note that ξ_*, η_* are defined on Ω_1^+, Ω_2^+ so that $D\xi_*$, $D\eta_*$ are in A) and so $dP_*(\delta\xi, \delta\eta) \in A$. Thus the diagrams (A4.12) hold with A' replaced by A. We therefore have the following equations:

(i) $dT_* = dP_* \circ dC_*$

(ii) $dT_* \circ dP_* = dP_* \circ dC_*$ (A4.14)

(iii) $dT_* \circ dP_* = dT_* = dP_* \circ dT_*$

holding on A. This technical problem over, we continue as in Ostlund et al (1983). We denote by Z_* the pair of functions $(\xi_*(x) - D\xi_*(x).x, \eta_*(x) - D\eta_*(x).x)$. Now it is a simple calculation (using the fixed point equations (A4.2)) to check that $dC_*Z_* = Z_*$, so that Z_* is an eigenvector of dC_* with eigenvalue one. The derivative of C is given by (applying the results of Appendix 3, Section A3.2)

$$dC(\xi, \eta)(\delta\xi, \delta\eta) = (\beta_*^{-1}\delta\eta(\beta_*x), \beta_*^{-1}\delta\eta(\xi(\beta_*x)) + \beta_*^{-1}D\eta(\xi(\beta_*x)).\delta\xi(\beta_*x)) \quad (A4.15)$$

Therefore

$$\begin{aligned} dC_*Z_*(x) &= (\beta_*^{-1}\eta_*(\beta_*x) - \beta_*^{-1}D\eta_*(\beta_*x).\beta_*\cdot x, \\ &\quad \beta_*^{-1}\eta_*(\xi_*(\beta_*x)) - \beta_*^{-1}D\eta_*(\xi_*(\beta_*x)).\xi_*(\beta_*x)) + \\ &\quad \beta_*^{-1}D^{\eta}{}_*(\xi_*(\beta_*x))(\xi_*(\beta_*x) - D\xi_*(\beta_*x).\beta_*.x) \\ &= (\beta_*^{-1}\eta_*(\beta_*x) - D\eta_*(\beta_*x).x, \\ &\quad \beta_*^{-1}\eta_*(\xi_*(\beta_*x)) - \beta_*^{-1}D\eta_*(\xi_*(\beta_*x)).D\xi_*(\beta_*x).\beta_*.x) \\ &= (\beta_*^{-1}\eta_*(\beta_*x) - D(\beta_*^{-1}\eta_*(\beta_*x)).x, \\ &\quad \beta_*^{-1}\eta_*(\xi_*(\beta_*x)) - D(\beta_*^{-1}\eta_*(\xi_*(\beta_*x))).x) \\ &= Z_* \end{aligned}$$

Now let $Z = (\delta\xi, \delta\eta)$ be a tangent vector in A. Then, from (A4.13),

$$dP_*Z = 0 \text{ iff } Z - Z_*\delta\lambda = 0 \text{ iff } Z = \delta\lambda.Z_*$$

where $\delta\lambda = a.\delta\xi(0) - b.\delta\eta(0)$. This means that Z_* is the only eigendirection of dP_* with eigenvalue 0. From (A4.14) (i) we see that $dT_*Z_* = 0$. We note that (A4.14) (ii) implies for $r > 1$

$$dP_* \circ dC_*{}^r = dT_*{}^r \circ dP_*$$

and

$$dP_* \circ (dC_* - \lambda I)^r = (dT_* - \lambda I)^r \circ dP_*. \quad (A4.16)$$

Now let λ be an eigenvalue of dC_* with generalised eigenvector Z i.e. there is $r \geq 1$ such that $(dC_* - \lambda)^r Z = 0$ and suppose $Z \neq kZ_*$ for any $k \in \mathbb{R}$. Applying dP_* and using (A4.16) we obtain $(dT_* - \lambda I)^r dP_* Z = 0$. Since $Z \neq kZ_*$, $dP_* Z \neq 0$ so that $dP_* Z$ is a generalised eigenvector of dT_* with eigenvalue λ. Conversely, let $Z \neq kZ_*$ be a generalised eigenvector of dT_* with eigenvalue $\lambda \neq 0$. Then $(dT_* - \lambda I)^r Z = 0$. Expanding this out gives

$$Z = -(-\lambda)^{-r}\left(\sum_{i=1}^{r} C_i^r (-\lambda)^{r-i} dT_*^{\,i}\right)Z \ .$$

Applying dP_* and using (A4.14)(iii) we see that $dP_* Z = Z$ and $(dT_* - \lambda I)^r dP_* Z = 0$. Then from (A4.16) we get $dP_*(dC_* - \lambda I)^r Z = 0$ and therefore $(dC_* - \lambda I)^r Z = cZ_*$, for some constant c. Now if $\lambda \neq 1$, then $\tilde{Z} = Z - cZ_*/(1 - \lambda)^r$ satisfies

$$(dT_* - \lambda I)^r \tilde{Z} = cZ_* - c/(1-\lambda)^r \ . \ (dC_* - \lambda I)^r Z_* = 0$$

since $dC_* Z_* = Z_*$. Now if $\lambda = 1$, then there are two possibilities. Either $c = 0$ in which case $(dC_* - \lambda I)^r Z = 0$, or $c \neq 0$, in which case $(dC_* - \lambda I)^r Z \neq 0$ but $(dC_* - \lambda I)^{r+1} Z = (dC_* - I)cZ_* = 0$, since Z_* is an eigenvector of dC_* with eigenvalue 1, so that Z is also a generalised eigenvector of dC_* with eigenvalue λ. This proves (iii).

□

<u>Proposition A4.3.</u> Let (ξ_*, η_*) be a fixed point of C on Ω_1, Ω_2 and let $\lambda \neq 0$. Then the pair of maps (ξ_*', η_*'), defined on the sets

$$\Omega_1' = \lambda^{-1}.\Omega_1, \ \Omega_2' = \lambda^{-1}.\Omega_2$$

respectively and given by

$$\xi_*'(x) = \lambda^{-1}.\xi_*(\lambda x), \qquad \eta_*'(x) = \lambda^{-1}.\eta_*(\lambda x)$$

is also a fixed point of C. Moreover, $dC_* = dC(\xi_*, \eta_*)$ and $dC_*' = dC(\xi_*', \eta_*')$ have identical eigenvalues complete with multiplicities.

<u>proof</u> This follows immediately since the map $S: A(\Omega_1) \times A(\Omega_2) \dashrightarrow A(\Omega_1') \times A(\Omega_2')$ given by

$$S(\xi, \eta)(x) = (\lambda^{-1}\xi(\lambda x), \lambda^{-1}\eta(\lambda x)) \qquad (A4.17)$$

is a differentiable isometric isomorphism with the property that $C' \circ S = S \circ C$ and $dC'_* \circ dS_* = dS_* \circ dC_*$ where $dS_* = dS(\xi_*, \eta_*)$ is an invertible operator.

□

Proposition A4.4 Let (ξ_*, η_*) be a fixed point of $T_{a,b}$ on Ω_1, Ω_2 (a,b not both zero) and let c, d (c, d not both zero) be given. Let S be defined as in equation (A4.17) with $\lambda = c.\xi(0) + d.\eta(0)$. Then $(\xi'_*, \eta'_*) = S(\xi_*, \eta_*)$ is a fixed point of $T_{c,d}$ defined on $\Omega'_1 = \lambda^{-1}\Omega_1$, $\Omega'_2 = \lambda^{-1}\Omega_2$ and $dT_{a,b}(\xi_*, \eta_*)$ and $dT_{c,d}(\xi'_*, \eta'_*)$ have identical eigenvalues complete with multiplicities.

proof By Proposition A4.3, dC_* and dC'_* have isomorphic spectral properties and by Proposition A4.2 dC_* and $dT_{a,b}$ and dC'_* and $dT_{c.d}$ only differ by a single eigenvalue 1. Hence $dT_{a,b}$ and $dT_{c,d}$ have identical eigenvalues complete with multiplicities.

□

This proposition shows that the precise choice of normalisation condition is not important. We are therefore justified in working in the (computationally) advantagous normalisation (A4.1) instead of the more natural condition (2.13)(i).

In view of this, we shall restrict discussion to the renormalisation transformation T as defined by equation (3.1).

A4.2 Properties of (ξ_*, η_*)

We shall make the following assumptions for the rest of this Appendix. Let T denote the renormalisation transformation with normalisation given by equation (A4.1).

Assumptions:

(i) (ξ_*, η_*) is an analytic fixed point of T defined on domains Ω_1, Ω_2 in $\mathbb{C}$ which satisfies equation (A4.9).

(ii) $-1 < \beta_* < 0$

(iii) $0 \in \Omega_1 \cap \Omega_2$, $[0, 1] \subseteq \Omega_2$.

(iv) $D\xi_*(0) = D\eta_*(0) = D^2\xi_*(0) = D^2\eta_*(0) = 0$, $D^3\xi_*(0)$, $D^3\eta_*(0) \neq 0$.

(v) $\kappa(\beta_*^2) = \beta_*^2$, $D\kappa(\beta_*^2) < 0$, $\kappa = \beta_*^{-1}\eta_*$.

With (i)-(iv) one may show that ξ_*, η_* are both analytic functions of x^3. This explains why we are justified in working in a space of analytic functions of x^3 when looking for a fixed point.

Proposition A4.5 Let (i)-(iv) be satisfied. Then ξ_*, η_* are both analytic functions of x^3.

proof(Ostlund et al (1983)) Let $\beta = \beta_*$ and let $\kappa(x) = \beta^{-1}\eta_*(x)$. Then κ is defined on Ω_2, $\kappa(0) = 1$ and, on a neighbourhood of 0, κ satisfies the equation

$$\kappa(x) = \beta^{-1}.\kappa(\kappa(\beta^2 x)) \qquad (A4.18)$$

Differentiating (A4.18) three times and evaluating at 0, we obtain

$$D\kappa(1) = \beta^{-5}. \qquad (A4.19)$$

Now assume that $D^p\kappa(0) = 0$ whenever $p < 3n$ and p is not a multiple of 3. Let q be such that $3n < q < 3(n + 1)$. Differentiating (A4.18) q times and evaluating at 0 one obtains (since all the other terms are zero by hypothesis)

$$D^q\kappa(0) = \beta^{2q-1}.D\kappa(1).D^q\kappa(0) \qquad (A4.20)$$

Substituting from equation (A4.19) into (A4.20) we see that $D^q\kappa(0) = 0$. Thus the Taylor expansion of κ about 0 contains only

terms of degree a power of three. Hence κ is an analytic function of x^3. Now $\xi_*(x) = \kappa(\beta x)$ and $\eta_*(x) = \beta\kappa(x)$ so that the same is true of both ξ_* and η_*.

□

From (i)-(v) Nauenberg has shown that ξ_* and η_* commute on a neighbourhood of 0.

Proposition A4.6. Let (i)-(v) be satisfied. Then ξ_* and η_* commute on a neighbourhood of 0, that is $\xi_* \circ \eta_* = \eta_* \circ \xi_*$.

proof (Nauenberg (1982)) As in the proof of Proposition A4.5 we let $\beta = \beta_*$ and $\kappa(x) = \beta^{-1}\eta_*(x)$. Then κ is defined on Ω_2, $\kappa(0) = 1$, and, on a neighbourhod of 0, satisfies equation (A4.18). We shall show that

$$\kappa(x) = \beta^{-2}\kappa(\beta^2\kappa(\beta x)). \qquad (A4.21)$$

Now $\xi_*(x) = \kappa(\beta x)$ and $\eta_*(x) = \beta\kappa(x)$, so that from equations (A4.18) and (A4.21) we obtain

$$\beta^{-2}.\eta_*(\xi_*(\beta x)) = \beta^{-2}\xi_*(\eta_*(\beta x))$$

so that ξ_* and η_* commute on a neighbourhood of 0 if (A4.21) is satisfied. We now prove (A4.21). We define $\tilde{\kappa}(x)$ to be the right hand side of (A4.21) i.e.

$$\tilde{\kappa}(x) = \beta^{-2}\kappa(\beta^2\kappa(\beta x)). \qquad (A4.22)$$

From equations (A4.21) and (A4.18) we obtain

$$\kappa(\beta^2\tilde{\kappa}(x)) = \kappa(\kappa(\beta^2\kappa(\beta x))) = \beta\kappa(\kappa(\beta x)) = \beta^2\kappa(\beta^{-1}x)$$

or

$$\kappa(x) = \beta^{-2}\kappa(\beta^2\tilde{\kappa}(\beta x)). \qquad (A4.23)$$

Now from equations (A4.22), (A4.23) and (A4.18) we obtain

$$\beta^2\tilde{\kappa}(\beta^{-1}x) = \kappa(\beta^2\kappa(x)) = \kappa(\kappa(\beta^2\tilde{\kappa}(\beta x))) = \beta\kappa(\tilde{\kappa}(\beta x))$$

which gives

$$\tilde{\kappa}(x) = \beta^{-1}.\kappa(\tilde{\kappa}(\beta^2 x)). \qquad (A4.24)$$

We note that $\kappa = \tilde{\kappa}$ is a solution of this equation, as it then becomes equation (A4.18). Now by Proposition A4.5, κ, and hence $\tilde{\kappa}$, is an

analytic function of x^3. Thus we may write

$$\kappa(x) = 1 + \sum_{i=0}^{\infty} C_i.x^{3i} \text{ and } \tilde{\kappa}(x) = \bar{C}_0 + \sum_{i=0}^{\infty} \bar{C}_i.x^{3i}$$

Now

$$\bar{C}_0 = \tilde{\kappa}(0) = \beta^{-2}.\kappa(\beta^2)$$

and by (v) $\kappa(\beta^2) = \beta^2$ so $\bar{C}_0 = 1$. Now κ is analytic at $x = 1$ so that equation (A4.24) can be written in the form

$$\sum_{i=1}^{\infty} \bar{C}_i \left(1 - \beta^{(6i-1)}.D\kappa(1) \right) x^{3i}$$

$$= \beta^{-1} \sum_{j=2}^{\infty} D^j\kappa(1)/j! \cdot \left[\sum_{i=1}^{\infty} \bar{C}_i \beta^{6i} x^{3i} \right]^j$$

Equating powers of x^3, for each $i > 1$ we obtain a recurrence relation for $\bar{C}_i$ in terms of $\bar{C}_j$ for $j = 1, \ldots, i$. $\bar{C}_1$ is not determined since for $i = 1$, the coefficient of $\bar{C}_i$ on the left hand side of the above equation is 0 as $D\kappa(1) = \beta^{-5}$ by equation (A4.19). Now since κ satisfies equation (A4.24), its coefficients C_i also satisfy the same recurrence relation as $\bar{C}_i$ so that $\kappa(x) = \tilde{\kappa}(x)$ if we can show that $C_1 = \bar{C}_1$. Now differentiating (A4.22) three times and evaluating at $x = 0$, we obtain:

$$\bar{C}_1 = D\kappa(\beta^2).\beta^3.C_1 \qquad \text{(A4.25)}$$

However, differentiating (A4.18) at $x = 1$, and using (v) together with equation (A4.19) gives

$$D\kappa(\beta^2) = \pm \beta^{-3}.$$

The plus sign must be taken as $\beta < 0$ (by (ii)) and $D\kappa(\beta^2) > 0$ (by (v)) so that (A4.25) reduces to $C_1 = \bar{C}_1$.

□

We note that (A4.18) evaluated at 0 gives $\kappa(1) = \beta_*$ so that (A4.18) when evaluated at 1 gives

$$\kappa(\kappa(\beta_*^2)) = \beta_*^2 \qquad \text{(A4.26)}$$

A4.3 Eigenvalues of dC_*

In this section, we shall assume that (i)-(v) of section A4.2 are satisfied. We shall also make use of Propositions A4.5 and A4.6.

Lemma A4.7

(a) $D\eta_*(\xi_*(0)) = \beta_*^{-4}$

(b) $D\xi_*(\eta_*(0)) = \beta_*^{-2}$

proof(Ostlund et al (1983)) The fixed point equations are

$$\xi_*(x) = \beta_*^{-1}\eta_*(\beta_* x) \tag{A4.27}$$

$$\eta_*(x) = \beta_*^{-1}\eta_*(\xi_*(\beta_* x)). \tag{A4.28}$$

Differentiating (A4.27) 3 times at 0, we get

$$D^3\xi_*(0) = \beta_*^2 D^3\eta_*(0) \tag{A4.29}$$

Differentiating (A4.28) 3 times at 0 and using $D\xi_*(0) = D^2\xi_*(0) = 0$, we get

$$D^3\xi_*(0) = \beta_*^2 D\eta_*(\xi_*(0)) D^3\xi_*(0)$$

Combining these two and using $D^3\eta_*(0) \neq 0$ gives

$$D\eta_*(\xi_*(0)) = \beta_*^{-4} \tag{A4.30}$$

This proves (a). Now from Proposition A4.6, we know that

$$\eta_*(\xi_*(x)) = \xi_*(\eta_*(x)) \tag{A4.31}$$

for x near 0. Differentiating (A4.31) 3 times at 0 we get

$$D\eta_*(\xi_*(0)).D^3\xi_*(0) = D\xi_*(\eta_*(0)) D^3\eta_*(0) \tag{A4.32}$$

Now since $D^3\eta_*(0) \neq 0$, (A4.29) together with (A4.32) give (c).

□

Lemma A4.8 Let q be a positive integer that is not a multiple of 3. Let $r > 0$ and let $Z = (\delta\xi, \delta\eta)$ be a tangent vector at (ξ_*, η_*). Let $(\delta\xi', \delta\eta') = dC_*^r Z$. Then

$$\begin{bmatrix} D^q\delta\xi'(0) \\ D^q\delta\eta'(0) \end{bmatrix} = \begin{bmatrix} 0 & \beta_*^{q-1} \\ \beta_*^{q-5} & 0 \end{bmatrix}^r \begin{bmatrix} D^q\delta\xi(0) \\ D^q\delta\eta(0) \end{bmatrix}$$

proof The case r = 0 is trivial. For r = 1 we have (from (A4.15))

$$\delta\xi'(x) = \beta_*^{-1}\delta\eta(\beta_* x)$$

$$\delta\eta'(x) = \beta_*^{-1}\delta\eta(\xi_*(\beta_* x)) + \beta_*^{-1}D\eta_*(\xi_*(\beta_* x)).\delta\xi(\beta_* x)$$

Differentiating q times at 0 and using the fact that ξ_*, η_* are analytic functions of x^3, we obtain

$$D^q\delta\xi'(0) = \beta_*^{q-1}D^q\delta\eta(0)$$

$$D^q\delta\eta'(0) = \beta_*^{q-1}D\eta_*(\xi_*(0)).D^q\delta\xi(0)$$

The conclusion of the lemma follows immediately from Lemma A4.7 (a). The case r > 1 follows by induction.

□

The following is a result of Ostlund et al (1983).

Proposition A4.9 Let $Z = (\delta\xi, \delta\eta)$ be a generalised eigenvector of dC_* with eigenvalue λ. Let one of $\delta\xi$, $\delta\eta$ not be an analytic function of x^3 and let q be the smallest integer, not a multiple of 3, for which one of $D^q\delta\xi(0)$, $D^q\delta\eta(0)$ is non-zero.

Then

(a) $\lambda = \pm\beta_*^{q-3}$

(b) Both $D^q\delta\xi(0)$ and $D^q\delta\eta(0)$ are non zero.

proof Let $r \geq 1$ be such that $(dC_* - \lambda I)^r Z = 0$. Then expanding this product, differentiating q times at 0, we obtain from Lemma A4.8

$$(M - \lambda I)^q \begin{bmatrix} D^q\delta\xi(0) \\ \\ D^q\delta\eta(0) \end{bmatrix} = 0, \quad M = \begin{bmatrix} 0 & \beta_*^{q-1} \\ \\ \beta_*^{q-5} & 0 \end{bmatrix} \qquad (A4.32)$$

Since the vector $D^qZ(0) \neq 0$, λ must be an eigenvalue of M. The eigenvalues of M are easily seen to be $\lambda = \pm\beta_*^{q-3}$. This proves (a). Note that the eigenvectors of M are not the basis vectors (1, 0) or (0, 1) so that both of $D^q\delta\xi(0)$ and $D^q\delta\eta(0)$ are non-zero. This proves (b).

□

We denote by $F(\xi, \eta)$ the map $\xi \circ \eta - \eta \circ \xi$. This map is well defined on a neighbourhood of 0 for all pairs (ξ, η) close to (ξ_*, η_*). Let $F^{(q)}$ denote the map $F^{(q)}(\xi, \eta) = D^q F(\xi, \eta)(0)$.

Lemma A4.10 Let $Z = (\delta\xi, \delta\eta)$ be a tangent vector at (ξ_*, η_*), let $q \geq 1$ not be a multiple of 3. Then if Z is tangential to $F^{(q)}(\xi, \eta) = 0$ (i.e. $d(F^{(q)})(\xi_*, \eta_*)Z = 0$) then

$$D\xi_*(\eta_*(0)).D^q\delta\eta(0) = D\eta_*(\xi_*(0)).D^q\delta\xi(0)$$

proof It is easy to see that since the operation of differentiation at 0 is linear we have that $d(F^{(q)})(\xi_*, \eta_*)Z = D^q(dF(\xi_*, \eta_*)Z)(0)$. Now

$$dF(\xi_*, \eta_*)Z(x) = \delta\xi(\eta_*(x)) + D\xi_*(\eta_*(x)).\delta\eta(x) - \delta\eta(\xi_*(x)) - D\eta_*(\xi_*(x)).\delta\xi(x) \qquad (A4.33)$$

Differentiating this q times at 0, and using the fact that ξ_*, η_* are analytic functions of x^3, we see the only terms remaining are

$$D\xi_*(\eta_*(0)).D^q\delta\eta(0) - D\eta_*(\xi_*(0)).D^q\delta\xi(0)$$

which gives the result.

□

Proposition A4.11

(a) For $q \geq 0$, the set $G^{(q)} = \{ (\xi, \eta) : F^{(q)}(\xi, \eta) = 0 \}$ is, in a neighbourhood of (ξ_*, η_*), a codimension one (Banach) submanifold of $A(\Omega_1) \times A(\Omega_2)$ containing (ξ_*, η_*).

(b) For $q \geq 0$, the set $A^{(q)} = \{ (\xi, \eta) : D^q\xi(0) = D^q\eta(0) = 0 \}$ is a codimension two submanifold of $A(\Omega_1) \times A(\Omega_2)$ containing (ξ_*, η_*).

proof

(a) We note that for (ξ, η) close to (ξ_*, η_*), F is well defined around 0 and hence $F^{(q)}$ is well defined. $(\xi_*, \eta_*) \in G^{(q)}$ by Proposition A4.6. We show that $d(F^{(q)})(\xi_*, \eta_*)$ is a surjective map. Then it will follow that $G^{(q)}$ is a codimension one submanifold from the implicit function theorem (see e.g. Lang (1962)). Now let $Z = (\delta\xi, \delta\eta)$ satisfy $D^i\delta\xi(0) = 0$, $i = 0, \ldots, q-1$, and $D^i\delta\eta(0) = 0$, $i = 0, \ldots, q$, $D^q\delta\xi(0) \neq 0$. Then

differentiating (A4.33) q times at 0, and using Lemma A4.7 (a) we get $d(F^{(q)})(\xi_*, \eta_*)Z = D\eta_*(\xi_*(0)).D^q\delta\xi(0) = -\beta_*^{-4}.D^q\delta\xi(0) \neq 0$. Therefore $d(F^{(q)})$ is surjective.

(b) This is immediate from the fact that $A^{(q)}$ is a codimension two closed linear subspace of $A(\Omega_1) \times A(\Omega_2)$.

□

Proposition A4.12 Let T be defined on an open neighbourhood of (ξ_*, η_*) in $A(\Omega_1) \times A(\Omega_2)$. Then:

(a) If (ξ, η) are both analytic functions of x^3, then the same is true of $T(\xi, \eta)$.

(b) If $F(\xi, \eta) \equiv 0$, then $F(T(\xi, \eta)) \equiv 0$.

(c) If $F^{(i)}(\xi, \eta) = 0$ for $i = 0, \ldots, q$, then $F^{(i)}(T(\xi, \eta)) = 0$ for $i = 0, \ldots, q$

(d) Let $D^i\xi(0) = D^i\eta(0) = 0$ for $i = 1, 2$ (i not a multiple of 3) and let $(\xi_1, \eta_1) = T(\xi, \eta)$. Then $D^i\xi_1(0) = D^i\eta_1(0) = 0$ for $i = 1, 2$.

proof (a) is immediate from the formula for T. (b) follows from

$$F(T(\xi, \eta))(x) = \beta^{-1}(\eta(\eta(\xi(\beta x))) - \eta(\xi(\eta(\beta)) \qquad (A4.34)$$

which is valid on a neighbourhood of 0, for (ξ, η) sufficiently close to (ξ_*, η_*). Now let the hypothesis of (c) hold. The case $i = 0$ follows immediately from evaluating (A4.34) at 0. Now differentiating (A4.34) i times, and using the case $i = 0$ i.e. $\xi(\eta(0)) - \eta(\xi(0)) = 0$ we get an expression whose terms all contain a term of the form $D^j(\xi \circ \eta - \eta \circ \xi)(0)$ for some $j = 1, \ldots, i$. By hypothesis these will be all zero for each $i = 1, \ldots, r$. This proves (c). Now let the hypotheses of (d) hold. Then $D^i\xi_1(0) = \beta^{i-1}D^i\eta(0) = 0$ and

$$D^1\eta_1(0) = D\eta(\xi(0)).D\xi(0) = 0,$$

$$D^2\eta_2(0) = \beta.(D^2\eta(\xi(0)).(D\xi(0))^2 + D\eta(\xi(0)).D^2\xi(0)) = 0.$$

□

Proposition A4.13 Let $Z = (\delta\xi, \delta\eta)$ satisfy the hypothesis of Proposition A4.9. Futhermore, let Z be tangential to the manifold $G^{(q)}$ of Lemma A4.11 i.e. $d(F^{(q)})(\xi_*, \eta_*)Z = 0$. Then $\lambda = \beta_* q^{-3}$ and $D^q\delta\xi(0)$, $D^q\delta\eta(0)$ have the same signs.

proof As in the proof of Proposition A4.9, we have (A4.32). Since Z is tangential to $G^{(q)}$ we have by lemma A4.10 and Lemma A4.7 (a) and (b)

$$D^q\delta\eta(0) = \beta_*^{-2} D^q\delta\xi(0) \qquad (A4.35)$$

We know from Proposition A4.9 that $\lambda = \pm\beta_* q^{-3}$. Now the eigenvectors of M for $\lambda = \pm\beta_* q^{-3}$ are $(1, \pm\beta_*^{-2})$ respectively, so that (A4.35) implies that $\lambda = \beta_*^{-2}$. The fact that $D^q\delta\xi(0)$, $D^q\delta\eta(0)$ have the same signs follows from (A4.35).

□

A5. Stable and Unstable Manifolds

This appendix contains a very brief review of the mathematics of stable and unstable manifolds. The standard reference to this topic is Hirsch et al (1977). We assume a knowledge of the theory of Banach manifolds. The necessary theory is contained in Lang (1962).

Let E be a Banach Manifold. Let V be an open set in E, and let $T: V \dashrightarrow E$ be a C^∞ map. Let Φ be a fixed point of T, i.e. $T(\Phi) = \Phi$.

Definition A stable manifold for T at Φ is a smooth submanifold W^s of V with the properties:

(a) $T(W^s) \subseteq W^s$

(b) If $\psi \in W^s$, then $\lim_{j\to\infty} T^j(\psi) = \Phi$

(c) (Transversality Condition) For $\psi \in W^s$, the range of dT_ψ is not contained in the tangent space to W^s at $T(\psi)$.

Because of the transversality condition (c), a transverse intersection with the stable manifold remains transverse under application of T. The definition of an unstable manifold is complicated by the fact that T is not necessarily invertible.

Definition An unstable manifold for T at Φ is a smooth submanifold W^u of E (not necessarily contained in V) with the properties:

(a) $T(W^u \cap V) \supseteq W^u$

(b) If $\psi \in W^u$, then there is a sequence ψ_j converging to Φ such that $\psi = T^j(\psi_j)$

(c) For any $\psi \in W^u \cap V$, the tangential derivative of T along W^u does not vanish.

Definition Let T be a C^∞ map from an open set V of a Banach space E into E. Let $T(\Phi) = \Phi$. Then Φ is a hyperbolic fixed point of T is $dT(\Phi)$

has no spectral value of modulus 1 in the complex plane.

The Local Stable and Unstable Manifold Theorems (see Hirsch et al (1977)) state that local stable and unstable manifolds exist in a neighbourhood of a hyperbolic fixed point.

We know from Hartmann's Theorem (see e.g. Irwin (1980)) that it is possible find a continuous coordinate change about a hyperbolic fixed point so that T is linear in the new coordinates. We need a differentiable linearisation. There may be obstructions to a complete linearisation but it is possible to linearise T differentiably in one direction. The following theorem is from Collet, Eckmann and Lanford (1980).

Theorem A5.1 (Collet et al (1980), Theorem 6.3) Let T be a C^2 map from an open set in a Banach space into the Banach space. Let Φ be a fixed point of T. Assume that $dT(\Phi)$ has a single simple eigenvalue $s > 0$ and the rest of the spectrum is contained in the interior of the unit disc. Then there exists a C^1 diffeomorphism of $B_1 \times (-1, 1)$ (B_1 is the open unit ball in some Banach space) onto a neighbourhood V of Φ such that

(i) (0, 0) represents Φ;

(ii) $B_1 \times \{0\}$ represents $W^s \cap V$;

(iii) $\{0\} \times (-1, 1)$ represents $W^u \cap V$.

(iv) T takes the form

$$(x,y) \to (M(x,y),\ s.y)$$

where $M(0,y) = 0$, $\| D_x M(x,y) \| < \alpha < 1$ for $(x,y) \in B^1 \times (-1, 1)$.

□

A6. Various Formulae involving T and dT

This appendix contains various formulae involving the renormalisation transformation T and its derivative dT.

We shall assume that (ξ, η) are defined on domains

$$\Omega_1 = \{x : | x^3 - a_1 | < r_1 \}$$

$$\Omega_2 = \{x : | x^3 - a_2 | < r_2 \}$$

with $0 \in \Omega_1 \cap \Omega_2$

and that there is an open set V of $A(\Omega_1) \times A(\Omega_2)$ on which (in the notation of Appendix 4) for $(\xi, \eta) \in V$, $\beta = \eta(0)$,

$$\beta.\Omega_1 < \Omega_2, \ \beta.\Omega_2 < \Omega_1, \ \xi(\beta.\Omega_2) < \Omega_2$$

A6.1 Formulae in terms of (ξ, η)

The renormalisation transformation T operates on pairs of maps $(\xi, \eta) \in V$:

$$T : \begin{bmatrix} \xi \\ \eta \end{bmatrix} (x) \quad \longrightarrow \quad \begin{bmatrix} \beta^{-1}\eta(\beta x) \\ \beta^{-1}\eta(\xi(\beta x)) \end{bmatrix} \qquad (A6.1)$$

where $\beta = \eta(0)$.

The derivative of T at a point $(\xi, \eta) \in V$, $dT(\xi, \eta)$, acting on tangent vectors $(\delta\xi, \delta\eta) \in A(\Omega_1) \times A(\Omega_2)$ is given by the formula

$$dT(\xi, \eta) \ : \begin{bmatrix} \delta\xi \\ \delta\eta \end{bmatrix} \longrightarrow \begin{bmatrix} \Delta\xi \\ \Delta\eta \end{bmatrix} \qquad (A6.2)$$

where

$$\Delta\xi(x) = (-\beta^{-2}\eta(\beta x) + \beta^{-1}D\eta(\beta x).x).\delta\beta + \beta^{-1}\delta\eta(\beta x)$$

$$\Delta\eta(x) = (-\beta^{-2}\eta(\xi(\beta x)) + \beta^{-1}D\eta(\xi(\beta x)).D\xi(\beta x).x).\delta\beta +$$

$$\beta^{-1}D\eta(\xi(\beta x)).\delta\xi(\beta x) + \beta^{-1}\delta\eta(\xi(\beta x))$$

and

$$\delta\beta = \delta\eta(0), \quad \beta = \eta(0).$$

This formula is obtained by application of the formulae in section A3.2. For $(\xi, \eta) \in V$, $dT(\xi, \eta)$ is an analyticity improving operator since $\Delta\xi$, $\Delta\eta$ are defined on larger domains than Ω_1, Ω_2,

We shall now assume that the existence of a fixed point (ξ_*, η_*) of T in V, and for which ξ_*, η_* are analytic functions of x^3, and with the property that on a neighbourhood of 0, $\xi_* \circ \eta_* = \eta_* \circ \xi_*$. We use the notation of Appendix 4:

$$F(\xi, \eta) = \xi \circ \eta - \eta \circ \xi$$

$$F^{(q)}(\xi, \eta) = D^q F(\xi, \eta)\ (0), \text{ for } q > 0$$

We consider the operators

$$dF_*^{(0)} = dF^{(0)}(\xi_*, \eta_*),\ dF_*^{(3)} = dF^{(3)}(\xi_*, \eta_*).$$

From the proof of Lemma A4.10 we know that

$$dF_*^{(q)}(\delta\xi, \delta\eta) = D^q(dF_*(\delta\xi, \delta\eta))(0)$$

where $dF_* = dF(\xi_*, \eta_*)$. From equation (A3.8) we get

$$dF_*(\delta\xi, \delta\eta) = \delta\xi(\eta_*(x)) + D\xi_*(\eta_*(x)).\delta\eta(x)$$
$$- \delta\eta(\xi_*(x)) - D\eta_*(\xi_*(x)).\delta\xi(x).$$

Using $\xi_*(0) = 1$, $\beta_* = \eta_*(0)$, we have the formulae

$$dF_*^{(0)}(\delta\xi, \delta\eta) = \delta\xi(\eta_*(0)) + D\xi_*(\eta_*(0)).\delta\eta(0)$$
$$- \delta\eta(\xi_*(0)) - D\eta_*(\xi_*(0)).\delta\xi(0)$$
$$= \delta\xi(\beta_*) + D\xi_*(\beta_*).\delta\eta(0)$$
$$- \delta\eta(1) - D\eta_*(1).\delta\xi(0) \qquad \text{(A6.3)}$$

and

$$dF_*^{(3)}(\delta\xi, \delta\eta) = D\delta\xi(\eta_*(0)).D^3\eta_*(0) + D^2\xi_*(\eta_*(0)).D^3\eta_*(0).\delta\eta(0)$$
$$+ D\xi_*(\eta_*(0)).D^3\delta\eta(0) - D\delta\eta(\xi_*(0)).D^3\xi_*(0)$$
$$- D^2\eta_*(\xi_*(0)).D^3\xi_*(0).\delta\xi(0) - D\eta_*(\xi_*(0)).D^3\delta\xi(0)$$
$$= D\delta\xi(\beta_*).D^3\eta_*(0) + D^2\xi_*(\beta_*).D^3\eta_*(0).\delta\eta(0)$$
$$+ D\xi_*(\beta_*).D^3\delta\eta(0) - D\delta\eta(1).D^3\xi_*(0)$$
$$- D^2\eta_*(1).D^3\xi_*(0).\delta\xi(0) - D\eta_*(1).D^3\delta\xi(0)$$

(A6.4)

(A6.4) uses the fact that ε_*, η_* are analytic functions of x^3 and so terms involving $D\varepsilon_*(0)$, $D\eta_*(0)$, $D^2\varepsilon_*(0)$ and $D^2\eta_*(0)$ are all 0.

These formulae simplify when $(\delta\xi, \delta\eta)$ is a eigenvector of dT_* with eigenvalue $\lambda \neq 0$. For, from (A6.2) we have

$$\lambda\delta\xi(0) = (-\beta_*^{-2}\eta_*(0) + \beta_*^{-1}D\eta_*(0).0).\delta\eta(0) + \beta_*^{-1}\delta\eta(0) = 0$$

where we have used $D\eta_*(0) = 0$ and $\eta_*(0) = \beta_*$. Hence $\delta\xi(0) = 0$ and equations (A6.3) and (A6.4) become

$$dF_*^{(0)}(\delta\xi, \delta\eta) = \delta\xi(\beta_*) + D\varepsilon_*(\beta_*).\delta\eta(0) - \delta\eta(1) \qquad \text{(A6.5)}$$

and

$$\begin{aligned} dF_*^{(3)}(\delta\xi, \delta\eta) = {} & D\delta\xi(\beta_*).D^3\eta_*(0) + D^2\varepsilon_*(\beta_*).D^3\eta_*(0).\delta\eta(0) \\ & + D\varepsilon_*(\beta_*).D^3\delta\eta(0) - D\delta\eta(1).D^3\varepsilon_*(0) \\ & - D\eta_*(1).D^3\delta\xi(0) \end{aligned} \qquad \text{(A6.6)}$$

A6.2 Formulae in terms of (h, k), $(\xi, \eta)(x) = (h, k)(x^3)$

We shall now restrict to the space $A^3(\Omega_1) \times A^3(\Omega_2)$ of pairs of analytic functions of x^3. The renormalisation transformation may be defined on the restriction of V to this space. $A^3(\Omega_1) \times A^3(\Omega_2)$ may be be identified with the space of analytic functions on $A(a_1, r_1) \times A(a_2, r_2)$ via the map $(\xi, \eta) \to (h, k)$, $\xi(x) = h(x^3)$, $\eta(x) = k(x^3)$. We make the convention that greek letters refer to functions defined on Ω_1, Ω_2, while roman letters refer to functions defined on $D(a_1, r_1)$, $D(a_2, r_2)$. We use the same notation for the maps T, F, $F^{(i)}$ that are induced by this identification. In terms of h, k the above formulae become

$$T : \begin{bmatrix} h \\ k \end{bmatrix}(x) \longrightarrow \begin{bmatrix} \beta^{-1}k(\beta^3 x) \\ \beta^{-1}k(h(\beta^3 x)^3) \end{bmatrix} \qquad \text{(A6.7)}$$

where $\beta = k(0)$.

The derivative of T at a point (h, k) $\in$ V, dT(h, k), acting on tangent vectors $(\delta h, \delta k) \in A(a_1, r_1) \times A(a_2, r_2)$ is given by the formula

$$dT(h, k) \; : \; \begin{bmatrix} \delta h \\ \\ \delta k \end{bmatrix} \; \longrightarrow \; \begin{bmatrix} \Delta h \\ \\ \Delta k \end{bmatrix} \qquad \text{(A6.8)}$$

where

$$\Delta h(x) = (-\beta^{-2} k(\beta^3 x) + 3\beta . Dk(\beta x).x\,).\delta\beta + \beta^{-1}\delta k(\beta^3 x)$$

$$\Delta k(x) = (-\beta^{-2} k(h(\beta^3 x)^3) + 9\beta . Dk(h(\beta^3 x)^3).h(\beta^3 x)^2 . Dh(\beta^3 x).x)\; \delta\beta$$

$$+ \beta^{-1} 3 . Dk(h(\beta^3 x)^3).h(\beta^3 x)^2 . \delta h(\beta^3 x) + \beta^{-1}\delta k(h(\beta^3 x)^3)$$

$$\delta\beta = \delta k(0), \quad \beta = k(0).$$

We now give the formulae for F, $F^{(q)}$. Note that if (ξ, η) are both analytic functions of x^3 then $F^{(q)}(\xi, \eta) = 0$ for q not a multiple of 3.

We have

$$F(h, k)\;(x) = h(k(x)^3) - k(h(x)^3)$$

and

$$dF(h, k)(\delta h, \delta k) = \delta h(k(x)^3) + 3Dh(k(x)^3).k(x)^2.\delta k(x)$$

$$- \delta k(h(x)^3) - 3Dk(h(x)^3).h(x)^2.\delta h(x)$$

Formulae (A6.3) and (A6.4) become

$$dF_*^{(0)}(\delta h, \delta k) = \delta h(k_*(0)^3) + 3Dh_*(k_*(0)^3).k_*(0)^2.\delta k(0)$$

$$- \delta k(h_*(0)^3) - 3Dk_*(h_*(0)^3).h_*(0)^2.\delta h(0)$$

$$= \delta h(\beta_*^3) + 3Dh_*(\beta_*^3).\beta_*^2.\delta k(0)$$

$$- \delta k(1) - 3Dk_*(1).\delta h(0) \qquad \text{(A6.9)}$$

and

$$dF_*^{(3)}(\delta h, \delta k) = 18(D\delta h(k_*(0)^3).k_*(0)^2.Dk_*(0)$$
$$+ 3D^2h_*(k_*(0)^3).k_*(0)^4.Dk_*(0).\delta k(0)$$
$$+ 2Dh_*(k_*(0)^3).k_*(0).Dk_*(0).\delta k(0)$$
$$+ Dh_*(k_*(0)^3).k_*(0)^2.D\delta k(0)$$
$$- D\delta k(h_*(0)^3).h_*(0)^2.Dk_*(0)$$
$$- 3D^2k_*(h_*(0)^3).h_*(0)^4.Dh_*(0).\delta h(0)$$
$$- 2Dk_*(h_*(0)^3).h_*(0).Dh_*(0).\delta h(0)$$
$$- Dk_*(h_*(0)^3).h_*(0)^2.D\delta h(0))$$
$$= 18(D\delta h(\beta_*^3).\beta_*^2.Dk_*(0)$$
$$+ 3D^2h_*(\beta_*^3).\beta_*^4.Dk_*(0).\delta k(0)$$
$$+ 2Dh_*(\beta_*^3).\beta_*.Dk_*(0).\delta k(0)$$
$$+ Dh_*(\beta_*^3).\beta_*^2.D\delta k(0)$$
$$- D\delta k(1).Dh_*(0)$$
$$- D^2k_*(1).Dh_*(0).\delta h(0)$$
$$- Dk_*(1).Dh_*(0).\delta h(0)$$
$$- Dk_*(1).D\delta h(0)) \qquad (A6.10)$$

When $(\delta h, \delta k)$ is an eigenvector of dT_*, we have $\delta h(0) = 0$ and equations (A6.9) and (A6.10) become

$$dF_*^{(0)}(\delta h, \delta k) = \delta h(\beta_*^3) + 3Dh_*(\beta_*^3).\beta_*^2.\delta k(0)$$
$$- \delta k(1) \qquad (A6.11)$$

and

$$dF_*^{(3)}(\delta h, \delta k) = 18(D\delta h(\beta_*^3).\beta_*^2.Dk_*(0)$$
$$+ 3D^2h_*(\beta_*^3).\beta_*^4.Dk_*(0).\delta k(0)$$
$$+ 2Dh_*(\beta_*^3).\beta_*.Dk_*(0).\delta k(0)$$
$$+ Dh_*(\beta_*^3).\beta_*^2.D\delta k(0)$$
$$- D\delta k(1).Dh_*(0)$$
$$- Dk_*(1).D\delta h(0)) \qquad (A6.12)$$

A7. Computer Program Listing and Explanatory Program Notes

A7.1 General Notes on the Computer Programs

1. The programs are written in the programming language C and have been run on Digital Equipment Corporation VAX-11/750 computers running UNIX at Warwick University and Queen Mary College, University of London. The Warwick Computer runs 4.1 Berkeley UNIX, while that at QMC runs version 4.2. A similar program written in Fortran and based on the computer program in Eckmann et al (1982) was run on the IBM 3081 computer at Cornell University, N.Y., U.S.A.

2. The UNIX operating system is now well known. The main reference is the Berkeley 4.1 and 4.2 UNIX manuals.

3. C was chosen for the programming for the following reasons:-

(a) C is the standard language for UNIX systems. Indeed, UNIX itself is writen in C. However, C is also now available under other operating systems.

(b) C has pleasant syntactical structures that make for easy and error free programming. However, C does not have some of the useful features of fortran such as the equivalence statement and we have had to simulate this construction using pointers (see 5.(a) below).

(c) C enables the user to define his/her own data structures. This is extremely useful for implementing the interval and function ball/vector operations discussed in Chapter 4 and Appendix 8. The method of using complex numbers to represent intervals that was used by Eckmann et al (1982) is sensible for fortran programming, but can be replaced by the more elegant structures in C. One feature of C that we have used is the ability to declare function subprograms that return structures as their values and to make

assignments between structures. These features are a newer development of C and our not contained in the description of C in Kernighan and Ritchie (1977). However they enable the programming to be greatly simplified, often eliminating the need to use to' temporary variables. Unfortunately they reduce program efficiency. C also has pointer or address manipulation facilities. While these can often increase the efficiency of a program, we have used them sparingly as they are sometimes confusing to the novice.

4. The standard reference for C is the book Kernighan and Ritchie (1977). However, as mentioned in the above paragraph, this is now out of date (and expensive!) so we have used the references Hogan (1984) and Wagner-Dobler (1985).

5. C has a number of constructs that are not standard programming. We mentioned these briefly.

(a) Pointers Pointers are address variables. For example,

```
double *p;
```

declares p to be a pointer to a variable of type double. *p then refers to the contents of the address of p. If x is a variable, then &x refers to the address of x. We have used pointers sparingly in the program. They are only used in the routines r_up, r_down and as a means of simulating the fortran equivalence statement.

(b) Structures C enables the programmer to define his/her own data structures. We have used the typedef statement to define new data types. Members of a structure are accessed by the "." operator. For example, the sequence of statements

```
typedef struct {
        double lo;   /* left  hand end-point of interval */
        double up;   /* right hand end-point of interval */
} interval;
```

```
. . . . . .
interval a;
double b;
. . . . . .
b = a.lo;
```

defines the data type "interval" to consist of two doubles lo and up, declares a to be of type interval, b to be of type double, and finally sets b to be the double lo of the interval a.

(c) Header Files and Macro Preprocessing Before compilation the C preprocessor is run. This enables the programmer to include separate files at compile time and to make macro substitutions. The files that are included are usually "header" files that contain declarations for the functions and variables that are used in the program but whose definitions are contained in another files. It is conventional in UNIX to suffix these files with ".h". The macros substitution facility of the C preprocessor is very useful. The substitutions can also include arguments as in the folowing example. A macro defintion

```
#define minus(x) -(x)
```

would result in the substitution of

```
y = -(1 + z);
```

for

```
y = minus(1 + z);
```

Notice that the brackets around x in the definition of minus are absolutely essential.

(d) Arrays The subsripts of an array declared to be of length N vary from 0 to N-1. Two dimensional arrays are stored in rows, rather than in columns as in fortran. C has the unusual syntax a[i][j] for a two dimensional array element.

(f) Functions All subprograms (unless explicitly declared "void") are

functions. By default they have type int. The arguments of a function are called by value rather than address as in fortran. The exceptions to this rule are arrays and character strings, where the address of the first element is passed to the function. This means that to change the value of an argument of a function, it is necesary to pass the address of the variable. This must be explicitly programmed in the function by making the argument a pointer variable. The versions of C that we use enable the programmer to define "structure valued" functions, i.e. functions that return a structure. This, while being rather inefficient since it requires a great deal of transferring to and from the stack, is extremely convenient from the programing point of view. We have managed to get rid of the large number of temporary storage variables required the fortran implementation of Eckmann et al (1982).

(g) Typecasting There an a simple device to converting one data type to another. The sequence:

```
int i;
double x;
. . . . . .
x = (double) i;
```

converts the int data type i to double and then assigns it to x. In fact this conversion would be implicit in the statement

```
x = i;
```

It is also worth mentioning that the variable type "register" is the same as "int" except that the compiler attempts to keep the variable easily accessible within the computer memory. This helps to increase efficiency.

(h) Extern statement This is the C version of the fortran "common" statement. A variable that is declared outside any of a subprogram

(including the subprogram "main" which is the main program) in a file is "global" to any of the subprograms in the file, although it may be redefined within any of these subprograms. These globally defined variables may be accessed from subprograms in another file using the "extern" statement. A variable within a subprogram may be declared static which means that it retains its value from one function call to the next.

(i) The following abbreviations are used in C:

```
i++      ≡ i = i + 1                    i--      ≡ i = i - 1

x += y   ≡ x = x + y                    x -= y   ≡ x = x - y

x *= y   ≡ x = x*y                      x /= y   ≡ x = x/y

( X ? Y : Z)  ≡ If X then Y else Z
```

7. There are two programs circ_prep and circ_proof. The main programs for these programs are contained in circ_prep.c and circ_proof.c respectively. The program circ_prep takes an approximate fixed point and calculates a basis change matrix (as described in Section 3.2) together with its inverse. The program circ_proof obtains the rigorous estimates required for the proof of Proposition 3.4. The two programs are independent. The actual data supplied to the program circ_proof is of no particular signficance. In fact, as remarked in Section 3.2, the choice of basis change matrix is ad hoc. The program circ_prep is included for the sake of completeness (although we have <u>not</u> included a program that calculates the approximate fixed point (as described in McKay (1982)) or one that calculates approximate eigenvalues and eigenvectors of the derivative at the fixed point). We shall only briefly describe the program circ_prep. The program circ_proof will be described in great detail.

A7.2 The program circ_prep

1. This program depends upon the folowing files:

circ_prep.c circ_nonrig.c r_function.c r_function.h

2. The files contain the following programs:

circ_prep.c	main, fnorm, eigit
circ_nonrig.c	r_t, r_dt
r_function.c	r_fzero, r_fnorm, r_fsmult, r_fscale, r_fsum, r_fdiff, r_fmult, r_fmult2, r_eval, r_fcomp, r_fcompl, r_fdcomp, r_fdcompl, r_fdkeval

The file r_function.h contains declarations for the functions in r_function.c.

3. The programs manipulate structures called real_functions. The definition of this structure is:

```
typedef  struct {
         double p[81];                    /*  polynomial part    */
         double a;                        /*  centre of domain   */
         double r;                        /*  radius of domain   */
} real_function
```

This structure represents the polynomial in x

$$f_p = \sum_{i=0}^{n} f.p[i].y^i,\ y = (x - f.a)/f.r$$

Here, n is an external parameter, set in circ_prep, which is the maximum degree of the polynomial f.p. We have n = 80. The file r_function.c contains routines that manipulate these real_functions.

(a) real_function r_fzero(a,r), double a,r, returns a real_function corresponding to the zero polynomial defined on the domain D(a, r).

(b) real_function r_fnorm(f), real_function f, returns the L^1-norm of f.

(c) real_function r_fscale(f,a,r), real_function f, double a,r, returns the real_function (f - a)/r.

(d) real_function r_fsum(f,g), real_function f,g, returns the real_function f + g.

(e) real_function r_fdiff(f,g), real_function f,g, returns the real_function f - g.

(f) real_function r_fmult(f,g), real_function f,g, returns the real_function f × g.

(g) real_function r_fmult2(f,g), real_function f,g, returns the real_function f × g, when g is at most linear.

(h) double r_eval(f,a), real_function f, double a, returns the value of f at x = a.

(i) real_function r_fcomp(f,g), real_function f,g, returns the real_function f o g.

(j) real_function r_fcompl(f,g), real_function f,g, returns the real_function f o g, when g is at most linear.

(k) real_function r_fdcomp(f,g), real_function f,g, returns the real_function Df o g.

(l) real_function r_fdcompl(f,g), real_function f,g, returns the real_function Df o g, when g is at most linear.

(m) double r_fdkeval(f,k,a), real_function f, int k, double a, returns the value of $D^k f$ at x = a, for k ⩾ 1.

4. The file circ_nonrig.c contains the following two routines:

(a) r_t(h,k,th,tk), real_function h, k, *th, *tk. This routine takes the real_functions h and k and returns the renormalised functions *th, *tk. The renormalisation process is straightforward to implement and it similar to that described in A7.3.

(b) r_dt(h,k,dtm), real_function h, k, double dtm[164][164]. This function returns in dtm the derivative matrix of dT at (h,k). The

rows and columns of dtm range from 0 to nk. The rows and columns nh, nk are not used - They correspond to the high order terms in the rigorous version of this routine (nh = n + 1, nk = 2n + 3). The method of calculation is similar to that of i_dt described in A7.3.

5. The file circ_prep.c contains the folowing routines:

(a) Main program. The program does the following:

(i) The program reads in the parameter n, sets up various dependent parameters and then reads in the aproximate fixed point (h, k).

(ii) The derivative matrix is calculated. First of all, the pair (h, k) is renormalised as this sets up various parameters used by r_dt e.g. the variable bet that corresponds to β. The routine r_dt is then called and the derivative matrix is returned in dtm.

(iii) The routine eigit is called to calculate the basis change matrix p and its approximate inverse pi.

(iv) The elements p[i][j], pi[i][j] for which none of i, j = nh, nk, are multiplied by 2.0 and 0.5 respectively. The helps to balance the nh, nk rows and columns of the rigourous version of this matrix, i_dtm.

(v) The matrices p, pi are written to the file ppi.

(b) fnorm. This function is used by eigit to estimate the norm of D (see (c) below). It also provides information on each column so that eigit may decide on which columns it is necessary to perform complex rotations.

(c) The routine eigit. This routine calculates the basis change matrix and its approximate inverse. The principle behind the routine is explained in section 3.2. It is an adaption of a program contained in Wilkinson and Reinsch (1971). A sequence of "complex rotations" (i.e.

rotations through a complex angle) on the basis elements are used to converge to the Jordan canonical form. We adapt this process in two ways:

(i) We only carry out the rotations when necessary. The routine scans the columns of the matrix and only performs rotations for those columns for which the off-diagonal elements are significant.

(ii) The process is terminated as soon as the matrix has the form required i.e. (3.3).

The reason for these changes is that we need to keep the basis change matrix as close to the identity as possible, while still transforming the matrix to the form (3.3). We could transform the matrix to its Jordan form but this is not necessary and increases the factor $\|p\|.\|p^{-1}\|$ which determines how error is transmitted under the basis change.

A7.3 The Program circ_proof

1. The program depends upon the following files:

circ_proof.c circ_rig.c i_function.c i_function.h i_routines.c i_routines.h r_routines.c r_routines.h rsp_routines.c r_macros.h r_up.c r_up.h print.c

2. The files contain the following programs:

circ_proof.c	main
circ_rig.c	i_t, i_dt, i_dteval
i_function.c	i_fzero, i_fconv, i_fnorm, r_ifnorm, i_fsmult, i_fscale, i_fsum, i_fdiff, i_fmult, i_fmult2, i_eval, i_fcomp, i_fcompl, i_fdcomp, i_fdcompl, i_fdkeval
i_routines.c	i_intr, i_intrl, i_inti, i_intil, r_av, i_neg, i_sum, i_prod, i_inv, i_quot, i_rmult, i_power, i_abs, r_abs, int_zero, i_matmult, i_matvec, r_matcol, i_matmult4, i_matvec4, r_matcol4, i_det4
r_routines.c	r_power_up, r_power_down, r_inv_up, r_inv_down
rsp_routines.c	r_sum_up, r_sum_down, r_prod_up, r_prod_down, r_diff_up, r_diff_down
r_up.c	r_up, r_down
print.c	print, prquot

3. The files i_function.h, i_routines.h, r_up.h contain typedef statements and declarations for the routines in i_function.c, i_routines.c and r_up.c respectively. The file r_macros.h contain macro definitions for the routines.c in rsp_routines.c. The file r_routines.h contains declarations for both the routines in

r_function.c and those in rsp_routines.c.

4. The routines in i_function.c, i_routines.c, rsp_routines.c, r_routines.c and r_up.c are described in detail in Appendix 8.

5. The file print.c contains routines connected with output of the results. There is a problem with decimal/binary conversion. The data are read in in decimal form but are converted to binary. This conversion is inexact. There are times that we wish to make print out exact (i.e. binary) results. The routine prquot prints out the contents of a double variable in the form

$$\text{sgn } a\ b/2^c,\ \text{sgn} = \pm,\ a,b,c \in \mathbb{N} \cup \{0\},$$

where the positive sign is omitted and the negative sign refers to the whole of a $b/2^c$, as is usual with fractions. The routine print is an extension of the C program printf that includes the new format %q corresponding to the routine prquot. This routine may be of interest to programmers as it is a function with a variable number arguments (like printf).

6. For the files circ_proof.c and circ_rig.c we have the folowing convention: Variables or functions that are "interval" in character are prefixed by "i_". Furthermore, often double variables and functions will be prefixed "r_".

7. In the following sections we shall often refer to variables and give formulae to which they correspond. The symbol "≃" should be loosely interpreted as "corresponds to".

8. We shall make the convention that $a_1, r_1, a_2, r_2 \in$ i_h.a, i_h.r,

i_k.a, i_k.r respectively, and stand for the centres and radii of the domains of i_h = h and i_k = k respectively.

9. The file circ_rig.c contains the following functions:

(a) i_t(i_h,i_k,i_th,i_tk)

interval_function i_h, i_k, *i_th, *i_tk;

This routine takes the pair (i_h, i_k) and returns the renormalised pair (*i_th, *i_tk). The calculations are straightforward. The variables are (in order of calculation):

i_h = h, i_k = k, (*i_th, *i_tk) = T(h, k)

i_bet = β = k(0), i_bet2 = β^2, i_bet3 = β^3, i_beti = β^{-1}

Note that i_zero is the zero interval [0, 0] (declared in i_routines.c).

i_bx1 = β^3.x written as a function on the domain of i_h i.e.

$\beta^3 x = r_1 * \beta^3 . y + \beta^3 a_1$, $y = (x - a_1)/r_1$

i_bx2 = β^3.x written as a function on the domain of i_k i.e.

$\beta^3 x = r_2 * \beta^3 . y + \beta^3 a_2$, $y = (x - a_2)/r_2$

i_kb = $k(\beta^3 x)$, *i_th = $\beta^{-1} . k(\beta^3 x)$

Note that in calculating i_kb the routine i_fcompl is used since i_bx1 is a linear function. A similar remark applies to i_hb below.

i_hb = $h(\beta^3 x)$, i_hb2 = $h(\beta^3 x)^2$, i_hb3 = $h(\beta^3 x)^3$

i_khb = $k(h(\beta^3 x)^3)$, *i_tk = $\beta^{-1} k(h(\beta^3 x)^3)$

(b) i_dt(i_h,i_k,i_dtm,err)

interval_function i_h, i_k; interval i_dtm[164][164];

double err[2][164];

This routine takes the pair (i_h, i_k) and returns the derivative of the renormalisation transformation in the matrices i_dtm and err. The routine calculates the derivative as a map of $\mathbb{R}^{n+1} \oplus \ell_1 \oplus \mathbb{R}^{n+1} \oplus \ell_1$ to itself, where the isomorphism is given by the basis $(e_i^{(1)}, 0)$, $(0, e_i^{(2)})$ where

$$e_i^{(1)} = ((x - a_1)/r_1)^i,\ i = 0, 1, \ldots$$

$$e_i^{(2)} = ((x - a_2)/r_2)^i,\ i = 0, 1, \ldots$$

the first (n+1) elements in each set being thought of as a basis for $\mathbb{R}^n$, the rest being thought of as a basis for ℓ_1 (see Appendix 3 for a discussion of the L^1-spaces). The derivative is calculated as the sum of the two matrices of operators

$$dT = M + E$$

where M is

$$\begin{bmatrix}
\alpha_{0,0} & \cdots & \alpha_{0,nh-1} & \beta_{0,nh} & \alpha_{0,nh+1}\cdots & \alpha_{0,nk-1} & \beta_{0,nk} \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
\alpha_{nh-1,0}\cdots & & \alpha_{nh-1,nh-1} & \beta_{nh-1,nh} & \alpha_{nh-1,nh+1} & \alpha_{nh-1,nk-1} & \beta_{nh-1,nk} \\
\gamma_{nh,0} & \cdots & \gamma_{nh,nh} & \varsigma_{nh,nh} & \gamma_{nh,nh+1}\cdots & \gamma_{nh,nk-1} & \varsigma_{nh,nk} \\
\alpha_{nh+1,0}\cdots & & \alpha_{nh+1,nh-1} & \beta_{nh+1,nh} & \alpha_{nh+1,nh+1} & \alpha_{nh+1,nk-1} & \beta_{nh+1,nk} \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
\alpha_{nk-1,0}\cdots & & \alpha_{nk-1,nh-1} & \beta_{nk-1,nh} & \alpha_{nk-1,nh+1} & \alpha_{nk-1,nk-1} & \beta_{nk-1,nk} \\
\gamma_{nk,0} & \cdots & \gamma_{nk,nh} & \varsigma_{nk,nh} & \gamma_{nk,nh+1}\cdots & \gamma_{nk,nk-1} & \varsigma_{nk,nk}
\end{bmatrix}$$

where $M \in \Lambda(\mathbb{R}^{n+1} \oplus \ell_1 \oplus \mathbb{R}^{n+1} \oplus \ell_1, \mathbb{R}^{n+1} \oplus \ell_1 \oplus \mathbb{R}^{n+1} \oplus \ell_1)$, $\alpha_{i,j} \in \mathbb{R}$, $\beta_{i,j} \in \Lambda(\ell_1, \mathbb{R}) = \ell_1^*$, $\gamma_{i,j} \in \Lambda(\mathbb{R}, \ell_1) \cong \ell_1$, $\varsigma_{i,j} \in \Lambda(\ell_1, \ell_1)$.

and E is

$$\begin{bmatrix}
\epsilon_0^{(1)} & \cdots & \epsilon_{nh-1}^{(1)} & \Phi_{nh}^{(1)} & \epsilon_{nh+1}^{(1)} & \cdots & \epsilon_{nk-1}^{(1)} & \Phi_{nk}^{(1)} \\
\epsilon_0^{(1)} & \cdots & \epsilon_{nh-1}^{(1)} & \Phi_{nh}^{(1)} & \epsilon_{nh+1}^{(1)} & \cdots & \epsilon_{nk-1}^{(1)} & \Phi_{nk}^{(1)}
\end{bmatrix}$$

where $E \in \Lambda(\mathbb{R}^{n+1} \oplus \ell_1 \oplus \mathbb{R}^{n+1} \oplus \ell_1, \ell_1 \oplus \ell_1)$, $\epsilon_j^{(i)} \in \Lambda(\mathbb{R}, \ell_1) \cong \ell_1$, $\Phi_j^{(i)} \in \Lambda(\ell_1, \ell_1)$.

The matrix M corresponds to the polynomial part and the high order function of the interval_function pair (i_dth, i_dtk). Information on the matrix M is returned in the interval matrix i_dtm. The matrix E corresponds to the error function in the interval_function pair (i_dth, i_dth). Information on this matrix is returned in the double matrix err. The following information is returned about the matrices

M and E. For every (h, k) contained within the interval_function pair (i_dth, i_dtk) we have

$$\alpha_{i,j} \in \text{i\_dtm[i][j]}, \ \|\beta_{i,j}\| \in \text{i\_dtm[i][j]},$$

$$\|\gamma_{i,j}\| \in \text{i\_dtm[i][j]}, \ \|\delta_{i,j}\| \in \text{i\_dtm[i][j]}.$$

The matrix elements i_dtm[i][j] for which one of i, j is equal to nh, nk are all symmetric intervals about 0.

For the matrix E we have $\|\epsilon_j^{(i)}\| \leq$ err[i][j], $\|\phi_j^{(i)}\| \leq$ err[i][j]. Notice that since in equation (A6.6) Δh is indpedent of δh, we have

$$\text{i\_dtm[i][j]} = [0, 0], \ 0 \leq i,j \leq nh$$

$$\text{err[0][j]} = 0, \ 0 \leq i \leq nh.$$

We therefore set the top left hand corner of i_dtm, err to zero at the begining of i dt.

The calculation of i_dtm and err is divided up into distinct sections

(1) Set up auxillary variables.

(2) Check that dT is a compact operator.

(3) Various auxillary calculations.

(4) Calculation of columns 0, ... n.

(5) Calculation of column nh.

(6) Calculation of columns nh + 1, ... nk - 1.

(7) Calculation of column nk.

We consider each of these sections in turn.

The following variables are declared to be as global in circ_rig.c and are common to i_t, i_dt, and i_dteval. They are set in i_t and so i_t must be called before calling i_dt.

```
interval i_bet, i_bet2, i_bet3, i_beti;

interval_function i_bx1, i_bx2, i_hb, i_hb2, i_hb3, i_kb, i_khb;
```

(1) Set up auxillary variables. We first of all set the top left hand corners of i_dtm are err to zero as mentioned above. We use the following

i_x1 = x written as a function on the domain of i_h i.e.

$x = r_1.y + a_1$, $y = (x - a_1)/r_1$.

i_x2 = x written as a function on the domain of i_k i.e.

$x = r_2.y + a_2$, $y = (x - a_2)/r_2$.

$i\_s0 = -a_2/r_2 = 0$ scaled to the domain of i_k.

i_three = 3

(2) Check that dT is a compact operator. We calculate the following variables

$i\_sbx1 = (\beta^3 x - a_2)/r_2$

This is the function i_bx1 scaled to the domain of i_k. rbx1 is an upper bound of the L^1-norm of i_sbx1.

$i\_sbx2 = (\beta^3 x - a_1)/r_1$

This is the function i_bx2 scaled to the domain of i_h. rbx2 is an upper bound of the L^1-norm of i_sbx2.

$i\_shb3 = (h(\beta^3 x)^3 - a_2)/r_2$

This is the function i_hb3 scaled to the domain of i_k. rhb3 is an upper bound of the L^1-norm of i_shb3.

The program checks that each of rbx1, rbx2, rhb3 is strictly less than 1. If this is not so then it prints an error message. For if these quantities are strictly less than 1, then (in the notation of Appendix 4)

$$\beta^3.D(a_1, r_1) < D(a_2, r_2)$$

$$\beta^3.D(a_2, r_2) < D(a_1, r_1)$$

$$h(\beta^3.D(a_2, r_2))^3 < D(a_2, r_2)$$

and dT(h, k) will be a compact operator. For then dT(h, k) is analyticity improving and so is compact by Proposition A3.6(b).

(3) Various auxillary calculations. The program calculates the following:

$$\text{i\_dkb} = Dk(\beta^3 x)$$

$$\text{i\_dkbx} = Dk(\beta^3 x).x$$

$$\text{i\_delb1} = 3\beta.Dk(\beta^3 x).x - \beta^{-2}k(\beta^3 x)$$

i_delb1 is the coefficient of $\delta\beta$ in the equation for Δh of (A6.8).

$$\text{i\_dkhb} = Dk(h(\beta^3 x)^3)$$

$$\text{i\_hdkhb} = h(\beta^3 x)^2.Dk(h(\beta^3 x)^3)$$

$$\text{i\_dhb} = Dh(\beta^3 x)$$

$$\text{i\_dhbx} = Dh(\beta^3 x).x$$

$$\text{i\_dkdh} = 9\beta.h(\beta^3 x)^2.Dk(h(\beta^3 x)^3).Dh(\beta^3 x).x$$

$$\text{i\_delb2} = 9\beta.h(\beta^3 x)^2.Dk(h(\beta^3 x)^3).Dh(\beta^3 x).x - \beta^{-2}k(h(\beta^3 x)^3)$$

i_delb2 is the coefficient of $\delta\beta$ in the equation for Δk of (A6.8)

(4) Calculation of columns 0, ..., n. i_dtk is the value Δk, as given in equation (A6.8) when the derivative acts on the basis tangent vector $\delta h = (e_j^{(1)}, 0)$, $\delta k = 0$, $j = 0, \ldots, n$. It is calculated using the following property of the basis $(e_j^{(1)}, 0)$. For $j = 0$

$$\text{i\_dtk} = 3\beta^{-1}h(\beta^3 x)^2.Dk(h(\beta^3 x)^3)$$

while for $j > 1$

$$\text{i\_dtk (for j)} = (\text{i\_dtk for } j - 1) \times \text{i\_sbx2}$$

For each j, the polynomial part of i_dtk form the rows nh + 1, .., nk - 1 of the jth column of i_dtm. The norm of the high order function of i_dtk is placed in i_dtm[nk][j] as the symmetric interval [-i_dtk.h, i_dtk.h]. The norm of the error function is placed in err[1][j].

(5) Calculation of column nh. This is a straightforward application of the formula for Δh in equation (A6.8) applied to the interval_function i_delh (= δh) defined by

i_delh.p[i] = [0, 0], i = 0, ..., n

i_delh.h = 1, i_delh.e = 0

i_delh.a = i_h.a, i_delh.r = i_h.r

This corresponds to a function ball containing all the basis elements $(e_j^{(1)}, 0)$, j = n + 1, n + 2, ... The following variables are calculated

i_delhb = $\delta h(\beta^3 x)$

i_dtk = $3\beta^{-1} h(\beta^3 x)^2 . Dk(h(\beta^3 x)^3) . \delta h(\beta^3 x)$

The polynomial part of i_dtk is placed in the rows nh + 1, ... nk - 1 of column nh (They are all symmetric intervals). The norm of the high order function is placed in i_dtm[nk][nh] as a symmetric interval [-i_dtk.h, i_dtk.h]. The error function is placed in err[1][nh].

(f) Calculation of columns nh + 1, ..., nk - 1. The interval_functions i_dth, i_dtk are respectively Δh, Δk, as given in equation (A6.8) when the derivative acts on the basis tangent vector $\delta h = 0$, $\delta k = e_\ell^{(2)}$, $\ell = 0, ..., n$. The following variables are used

i_delkb = $\beta^{-1} \delta k(\beta^3 x)$

i_dlkhb = $\beta^{-1} \delta k(h(\beta^3)^3 x)$

i_delb = $\delta\beta$

i_dth and i_dtk are calculated using the following property of the basis $(0, e_\ell^{(2)})$. For $\ell = 0$

i_delkb = β^{-1}

i_dlkhb = β^{-1}

i_delb = 1

while for $\ell > 1$

i_delkb for ℓ = (i_delkb for ℓ - 1) x i_sbx1

i_dlkhb for ℓ = (i_dlkhb for ℓ - 1) x i_shb3

i_delb for ℓ = (i_delb for ℓ - 1) x i_s0

Then for each ℓ the interval_functions i_dth, i_dtk are given by

$$i\_dth = i\_delkb + i\_delbl \times i\_delb$$

$$i\_dtk = i\_delkhb + i\_delb2 \times i\_delb$$

For each $j = nh + 1 + \ell$, the polynomial part of i_dth form the rows 0, ..., nh - 1 of the jth column of i_dtm. The norm of the high order function of i_dth is placed in i_dtm[nh][j] as the symmetric interval [-i_dth.h, i_dth.h]. The norm of the error function is placed in err[0][j]. The polynomial part of i_dtk form the rows nh + 1, ..., nk - 1 of the jth column of i_dtm. The norm of the high order function of i_dtk is placed in i_dtm[nk][j] as the symmetric interval [-i_dtk.h, i_dtk.h]. The norm of the error function is placed in err[1][j].

(7) Calculation of column nk. This is a straightforward application of the formula for Δk in equation (A6.8) applied to the interval_function i_delk (= δh) defined by

$$i\_delk.p[i] = [0, 0],\ i = 0, \ldots, n$$

$$i\_delk.h = 1,\ i\_delk.e = 0$$

$$i\_delk.a = i\_k.a,\ i\_delk.r = i\_k.r$$

This corresponds to a function ball containing all the basis elements $(0, e_\ell^{(2)})$, $\ell = n + 1, n + 2, \ldots$ The following variables are calculated

$$i\_delb = \delta\beta = \delta k(0)$$

$$i\_dth = \delta\beta \times i\_delbl + \beta^{-1}\delta k(\beta^3 x)$$

$$i\_delkb = \delta k(h(\beta^3 x)^3)$$

$$i\_dtk = \delta\beta \times i\_delb2 + \beta^{-1}\delta k(h(\beta^3 x)^3)$$

The polynomial part of i_dth is placed in the rows 0, ... nh - 1 of column nk (They are all symmetric intervals). The norm of the high order function is placed in i_dtm[nh][nk] as a symmetric interval [-i_dth.h, i_dth.h]. The error function norm is placed in err[0][nk].

The polynomial part of i_dtk is placed in the rows nh + 1, ... nk - 1 of column nk (They are all symmetric intervals). The norm of the high order function is placed in i_dtm[nk][nk] as a symmetric interval [-i_dtk.h, i_dtk.h]. The error function norm is placed in err[1][nk].

(c) i_dteval(i_delh, i_delk, i_dth i_dtk)

interval_function i_delh, i_delk, *i_dth, *i_dtk

This routine returns in (i_dth, i_dtk) the value of the derivative dT(h, k) acting on the pair (i_delh, i_delk). The routine uses various parameters that are set in i_t and i_dt so that these two routines must be called prior to calling i_dteval. The following variables that are declared globally in circ_rig.c are used in i_dteval:

interval_function i_bxl, i_bx2, i_hb3 (set in i_t)

interval i_beti (set in i_t)

interval_function i_delbl, i_delb2, i_hdkhb (set in i_dt)

The calculation of (i_dth, i_dtk) = (Δh, Δk) acting on (i_delh, i_delk) = (δh, δk) is a straightforward application of (A6.8). Note that i_delbl is the coefficient of δβ in the equation for Δh in (A6.8) and i_delb2 is the coefficient of δβ in the equation for Δk in (A6.8). The function i_hdkhb = $h(\beta^3 x)^2 . Dk(h(\beta^3 x)^3)$.

10. The file circ_proof.c contains the main program circ_proof. We make the following general points

(a) interval objects begin with the prefix "i_".

(b) The program manipulates a number of large matrices (164 x 164 interval matrices each requiring 430 336 bytes, and 164 x 164 double matrices each requiring half that amount). We have therefore contrived to keep the number of such matrices to a minimum. We have found that we may make do with four of these matrices, provided we use a simulation of the fortran equivalence statement. For example, the following declaration

```
interval i_api[164][164];
interval (*i_temp)[164] = i_api;
```

has the effect that i_api and i_temp are interval matrices that refer to the same storage location. Of course, if we one must be careful that the matrices are not used at the same time and are properly initialised each time they are used.

(c) There is a minor problem of binary/decimal conversion. The data that is fed into the program is stored in decimal. However, this is converted to binary by the computer. This conversion process is not exact and we must always make the caveat that decimal values are only approximations to the exact values of the variables to which we refer. The input data given for the program is thus only approximately the approximate fixed point with which we work! When we wish to make exact statements we shall print out the result in binary form (using the routine prquot).

(d) The proof involves finding estimates on matrices of operators on various L^1-spaces (see Appendix 3 for a description of L^1-spaces). In the program, we represent these matrices of operators as interval matrices. These interval matrices contain information on the elements

of the matrices of operators. To fix ideas, we consider a simple example. It is straightforward to extrapolate from this simple example to other situations. Consider the operator $B \in \Lambda(\mathbb{R}^{m+1} \oplus \ell_1, \mathbb{R}^{m+1} \oplus \ell_1)$

$$\begin{bmatrix} \alpha_{0,0} & \cdots & \alpha_{0,m} & \beta_0 \\ \vdots & & \vdots & \vdots \\ \alpha_{m,0} & \cdots & \alpha_{m,m} & \beta_m \\ \gamma_0 & \cdots & \gamma_m & \delta \end{bmatrix}$$

where $\alpha_{i,j} \in \mathbb{R}$, $\beta_i \in \Lambda(\ell_1, \mathbb{R}) = \ell_1^*$, $\gamma_j \in \Lambda(\mathbb{R}, \ell_1) \cong \ell_1$, $\delta \in \Lambda(\ell_1, \ell_1)$. We represent this matrix B by the interval matrix B:

$$\begin{bmatrix} A_{0,0} & \cdots & A_{0,m} & B_0 \\ \vdots & & \vdots & \vdots \\ A_{m,0} & \cdots & A_{m,m} & B_m \\ C_0 & \cdots & C_m & D \end{bmatrix}$$

where $m \geq 1$, $\alpha_{i,j} \in A_{i,j}$, $\|\beta_i\| \in B_i$, $\|\gamma_j\| \in C_j$, $\| \delta \| \in D$, and B_i, C_j, D are all <u>symmteric intervals.</u> Let u be a vector in $\mathbb{R}^{m+1} \oplus \ell_1$. We write:

$$\begin{bmatrix} u_0 \\ \vdots \\ u_m \\ v \end{bmatrix}$$

with $u_i \in \mathbb{R}$, $v \in \ell_1$. Analogously we represent u be the interval vector U

$$\begin{bmatrix} U_0 \\ \vdots \\ U_m \\ V \end{bmatrix}$$

where $u_i \in U_i$, $\| v \| \in V$ and V is a symmetric interval. We claim that the product $Bu \in \mathbb{R}^{m+1} \oplus \ell_1$ is represented by a vector formed by the interval arithmetic product B.U. To see this, we write $w = Bu$ as

$$\begin{bmatrix} w_0 \\ \vdots \\ w_m \\ y \end{bmatrix}$$

Then for $0 < i < m$.

$$w_i = \alpha_{i,0}.u_0 + \ldots + \alpha_{i,m}.u_m + \beta_i(v)$$

Now, $\|\beta_i(v)\| < \|\beta_i\|.\|v\|$ and so $\beta_i(v) \in [-\|\beta_i\|.\|v\|, \|\beta_i\|.\|v\|]$ and thus

$$w_i \in A_{i,0} \times_i U_0 +_i \ldots +_i A_{i,m} \times_i U_m + B_i \times_i V$$

since $B_i \times_i V$ is a symmetric interval containing $\|\beta_i\|.\|v\|$. Furthermore,

$$\begin{aligned} \|y\| &< \| \gamma_0.u_0 + \ldots + \gamma_m.u_m + \delta(v) \| \\ &< \|\gamma_0\|.|u_0| + \ldots + \|\gamma_m\|.|u_m| + \|\delta\|.\|v\| \\ &\in C_0 \times_i U_0 +_i \ldots +_i C_m \times_i U_m +_i D \times_i V \end{aligned}$$

Note that this last expression is a symmetric interval. This proves the claim.

11. The program may be broken down into the following stages:

(a) Read in the parameters, the approximate fixed point and initialise various dependent parameters.

(b) Read in the basis change matrix r_p and its approximate inverse r_api and find interval matrices i_p, i_pi containing r_p and its inverse.

(c) Estimate the accuracy of the approximate fixed point.

(d) Form a ball around the approximate fixed point (h0, k0).

(e) Calculate the derivative matrix.

(f) Convert the derivative matrix to the p-basis and contract to a 4 x 4 interval matrix.

(g) Check that the Newton map is a contraction on the ball around (h0, k0).

(h) Estimate β_* and check that $g(\beta_*^6)^3 = \beta_*^6$, and $Dg(\beta_*^6) < 0$ where $g = \beta_*^{-1}k$.

(i) Check that h, k have non-zero derivative at 0.

(j) Check that the spectrum of dT_* consists of three simple eigenvalues λ_0, λ_1, λ_2 with the rest of the spectrum contained completely within the disc D(0, 0.875).

(k) Obtain good bounds for λ_0, λ_1, λ_2 and their eigenvectors and show that for λ_1 and λ_2 their eigenvectors violate the commuting conditions $dF_*^{(0)}(\delta h, \delta k) = 0$, $dF_*^{(3)}(\delta h, \delta k) = 0$, respectively.

We give a detailed description of each of these stages.

(a) Read in the parameters, the approximate fixed point and initialise various dependent parameters.

n = degree of the polynomial part of interval_functions,

n = 80.

nh = n + 1, nh is the position of the h-high order terms in the derivative matrix i_dtm.

nk = 2n + 3, nk is the position of the k-high order terms in the derivative matrix i_dtm.

a1, r1, a2, r2 - the centres and radii of the domains of h, k respectively. The exact binary values are printed out to provide a precise statement of the domain of definition of the fixed point.

i_a1, i_r1, i_a2, i_r2 - interval versions of a1, r1, a2, r2.

h0, k0 - approximate fixed point (real_functions).

i_h0, i_k0 - interval_function versions of h0, k0.

(b) Read in the basis change matrix r_p and its approximate inverse r_api and find interval matrices i_p, i_pi containing r_p and

its inverse.

The basis change matrix P is of the form

$$\begin{bmatrix} P_{0,0} & \cdots & P_{0,nh-1} & 0 & & P_{0,nh+1}\cdots & P_{0,nk-1} & 0 \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & \vdots \\ P_{nh-1,0}\cdots & & P_{nh-1,nh-1} & 0 & & P_{nh-1,nh+1} & P_{nh-1,nk-1} & 0 \\ 0 & & 0 & I & & 0 & 0 & 0 \\ P_{nh+1,0}\cdots & & P_{nh+1,nh-1} & 0 & & P_{nh+1,nh+1} & P_{nh+1,nk-1} & 0 \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & \vdots \\ P_{nk-1,0}\cdots & & P_{nk-1,nh-1} & 0 & & P_{nk-1,nh+1} & P_{nk-1,nk-1} & 0 \\ 0 & & 0 & 0 & & 0 & 0 & I \end{bmatrix} \qquad \text{(A7.1)}$$

We again comment that the actual matrix consists of the binary version stored in the computer. The matrix is read in as r_p, a 164 × 164 matrix with 1 replacing the identity operator I of (A7.1), together with an approximate inverse r_api. Two things must be done. Firstly, we must show that the matrix r_p is invertible and obtain an interval matrix which contains the inverse. To do this we use a method contained in Moore (1966). We form the interval matrix product i_papi = i_p × i_api, where i_p, i_api are the interval versions of r_p, r_api respectively. Now i_papi will be close to the identity matrix I but not necessarily containing it. We write i_e for the matrix (i_papi - I). We note that the actual product r_p × r_api is contained in the matrix i_papi. Now we show that all matrices in i_e have L^1-norm less than 1. This is done by calculating e_1 = r_matcol(i_e, 0, nk, 0, nk). From this we conclude that the norm of the true matrix r_p × r_api - I is less than 1, so that r_p is invertible and that

$$r\_pi \times r\_api \in i\_papi = I + i\_e$$

and therefore

$$r\_p^{-1} \in i\_papi \times (I + i\_e)^{-1}$$

where $(I + i\_e)^{-1}$ means $\{(I + e)^{-1} | e \in i\_e\}$. However, for $e \in i\_e$

$$(I + e)^{-1} = I - e \times (I + e)^{-1}$$

so that

$$(I + e)^{-1} \in I - M$$

where M is an interval matrix with every element

$$(-\|e \times (1 + e)^{-1}\|, \ \|e \times (1 + e)^{-1}\|)$$

This matrix is estimated as $i\_e3 = I + M$, where M is an interval matrix with every element $i\_e2 \supseteq [-e1/(1 - e1), e1/(1 - e1)]$, and we have

$$r\_p^{-1} \in i\_pi = i\_api \times i\_e3.$$

Thus we obtain interval matrices i_p, i_pi containing respectively the basis change matrix and its inverse. We shall refer to the basis $(e_i^{(1)}, 0)$, $(0, e_i^{(2)})$ for $i = 0, 1, 2, \ldots$ as the standard basis and the basis to which we transform as the p-basis. The p-basis is defined as

$$f_i^{(1)} = e_i^{(1)}.P, \ f_i^{(2)} = e_i^{(2)}.P, \text{ for } i = 0, 1, \ldots.$$

The left multiplication is because P represents the transformation on coordinates rather than on the vectors themselves. Note that $f_i(1) = e_i(1)$, $f_i(2) = e_i(2)$ for $i \geq n+1$. We shall look on the space L expressed with respect to the p-basis as $\mathbb{R}^3 \oplus \ell_1$ with the first 3 basis vectors $f_0^{(1)}$, $f_1^{(1)}$, $f_2^{(1)}$ being though of as spanning $\mathbb{R}^3$ and the rest of the p-basis being thought of ℓ_1.

(c) Estimate the accuracy of the approximate fixed point. The approximate fixed point is renormalised using the routine i_t. The following variables are used:

i_h0, i_k0 - the interval versions of the approximate fixed point.

i_th0, i_tk0 - the renormalised approximate fixed point

i_dif0 - interval of length 0, ..., nk containing the difference

between (i_th0, i_tk0) and (i_h0, i_k0).

Note that i_th0.e = i_tk0.e = 0, because i_h0, i_k0 only have non-zero polynomial parts and the error terms come from this high order terms. The vector is converted to the p-basis by the matrix i_pi giving the vector i_difp. The approximate Newton map is defined to be:

$$N(h, k) = (h, k) - J(T(h, k) - (h, k))$$

where J is the diagonal matrix (with respect to the p-basis regarded as $\mathbb{R}^3 \oplus \ell_1$)

$$\begin{bmatrix} -1/3.234 & & & \\ & 1/1.239 & & \\ & & -1/2 & \\ & & & I \end{bmatrix}$$

or, more precisely, the binary approximation to this matrix. i_difp is multiplied by this matrix and thus i_difp is an interval vector representing N(h0, k0) - (h0, k0).

r_difp is the L^1-norm of i_difp.

(d) Form a ball around the approximate fixed point.

We form a ball about the approximate fixed point of radius ten times r_difp. Then the approximate Newton map N will be a contraction provided the norm of its derivative is less than 0.9. The ball around the approximate fixed point (h_0, k_0) is, of course, defined with respect to the L^1-norm of the p-basis. Of course, it would be foolish to express (h_0, k_0) in this basis, and then convert back, since this would only increase the amount of error. What we do is to form a ball about (h_0, k_0) with respect to the standard basis that must contain the ball around (h_0, k_0) with respect to the p-basis.

rad - radius of ball around (h0, k0) in the p-basis.

This ball will be contained in the pair of interval_functions i_h, i_k about h0, k0 defined by

i_h.p[i] = i_h0.p[i], i = 0, ..., n

i_h.h = 0, i_h.e $\rangle$ rnp1*rad

i_k.p[i] = i_k0.p[i], i = 0, ..., n

i_k.h = 0, i_k.e $\rangle$ rnp2*rad

where

$$\text{rnp1} \rangle \max \{ \sum_{i=0}^{nk} |p_{ij}| \ : \ 0 \langle j \langle nh \}$$

$$\text{rnp2} \rangle \max \{ \sum_{i=0}^{nk} |p_{ij}| \ : \ nh + 1 \langle j \langle nk \}$$

where r_p = (p_{ij}). For if we regard P as the matrix

[$P^{(1)}$, $P^{(2)}$], $P^{(i)} \in \Lambda(\ell_1, \ell_1 \oplus \ell_1)$ for i = 1 , 2

then rnp1 $\rangle$ $\|P^{(1)}\|$, rnp2 $\rangle$ $\|P^{(2)}\|$ and so bound the radii of balls around h, k respectively (each viewed as lying in ℓ_1 via the standard basis). (See Appendix 3 for a discussion of transmission of error under a basis change.)

(e) Calculate the derivative matrix.

The derivative matrix i_dtm and the associated error matrix err are calculated via the routine i_dt. Note that (i_h, i_k) are first renormalised to set up various parameters within circ_rig.c common to both i_t and i_dt. Note also that since i_h, i_k are both interval_function's that contain the ball around the approximate fixed point, the matrices i_dtm and err contain entries that are valid for <u>all</u> (h, k) in this ball.

(f) Convert the derivative matrix to the p-basis and contract to a 4 × 4 interval matrix.

Information on the derivative is returned from i_dt in the interval matrix i_dtm and the double matrix err as described in the notes for i_dt. These represent the derivative viewed as an operator on $\mathbb{R}^{n+1} \oplus \ell_1 \oplus \mathbb{R}^{n+1} \oplus \ell_1$ with respect to the standard basis. We look on the derivative as an operator on $\mathbb{R}^3 \oplus \ell_1$ where the isomorphism is given via the p-basis. We must therefore conjugate the derivative matrix with the basis change matrix P. With respect to the p-basis, the derivative is

$$P^{-1}\,(M + E)\,P \qquad \text{(A7.1)}$$

We regard this as an operator on $\mathbb{R}^3 \oplus \ell_1$ by "contracting down to a 4 × 4 matrix." This means that we regard (A7.1) as an operator matrix

$$\begin{bmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \beta_0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \beta_1 \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \beta_2 \\ \gamma_0 & \gamma_1 & \gamma_2 & \delta \end{bmatrix} \quad \in \quad \Lambda(\mathbb{R}^3 \oplus \ell_1,\ \mathbb{R}^3 \oplus \ell_1)$$

and we represent this as an interval matrix i_dtm4

$$\alpha_{i,j} \in \text{i\_dtm4}[i][j],\ 0 \leq i,j \leq 2,\ \|\beta_i\| \in \text{i\_dtm4}[i][3],\ 0 \leq i \leq 2$$

$$\|\gamma_j\| \in \text{i\_dtm4}[3][j],\ 0 \leq j \leq 2,\ \|\delta\| \in \text{i\_dtm4}[3][3].$$

The matrix elements i_dtm4[i][j] for which at least one of i, j = 3 are all symmetric intervals. Thus "contracting to a 4 × 4 matrix" requires discarding information about the derivative matrix in order to make the calculations more manageable.

The process is done in two stages. Firstly, the interval matrix i_dtm representing $P^{-1}.M.P$ is conjugated to the p-basis using the i_matmult routine. The conjugated matrix is again called i_dtm but this should not cause any confusion since we do not use the matrix with respect

to the standard basis. After conjugating i_dtm, the matrix is contracted to a 4 × 4 matrix i_dtm4 and the contribution from the matrix E is added. The contraction of i_dtm is simply a matter of viewing $P^{-1}.M.P$ as an operator on $\mathbb{R}^3 \oplus \ell_1$ by regarding $f_0^{(1)}$, $f1^{(1)}$, $f_2^{(1)}$ as spanning $\mathbb{R}^3$ and the rest of the matrix as ℓ_1. For example, δ is simply the submatrix of $P^{-1}.M.P$ formed from the rows 3 to nk, and columns 3 to nk. Finding a norm for δ is a matter of finding the maximum column sum norm of this submatrix. This is obtained by summing the norms of the elements of this submatrix. To calculate the contribution from $P^{-1}.E.P$, we evaluate the matrix E.P directly by calculating the double matrix errp given as

$$\text{errp[i][j]} = \Sigma \, \|\epsilon_k^{(i)}\| . |p_{k,j}| = \Sigma \, \text{err[i][k]}.\text{r\_abs(i\_p[k][j])}$$

where the sums range from 0 to nk. This gives bounds on the norms of E.P viewed as a matrix of operators in the same way as E. To calculate $P^{-1}.E.P$ we look on P^{-1} as the matrix of operators

$$\begin{bmatrix} P_{0,0}^{-1} & P_{0,1}^{-1} \\ P_{0,0}^{-1} & P_{0,1}^{-1} \\ P_{0,0}^{-1} & P_{0,1}^{-1} \\ P_{0,0}^{-1} & P_{0,1}^{-1} \end{bmatrix} \quad \epsilon \quad \Lambda(\ell_1 \oplus \ell_1, \mathbb{R}^3 \oplus \ell_1)$$

Then the contribution of $P^{-1}.E.P$ is determined by the maximum column sum of these submatrices stored in the double matrix rnpi

$$\text{rnpi[0][0]} = \max \{ \quad | p_{0j}^{-1}| : 0 < j < nh \}$$

$$\text{rnpi[1][0]} = \max \{ \quad | p_{1j}^{-1}| : 0 < j < nh \}$$

$$\text{rnpi[2][0]} = \max \{ \quad | p_{2j}^{-1}| : 0 < j < nh \}$$

$$\text{rnpi[3][0]} = \max \{ \Sigma \, | p_{ij}^{-1}| : 0 < j < nh \}$$

$$\text{rnpi[0][1]} = \max \{ \quad | p_{0j}^{-1}| : nh+1 < j < nk\}$$

rnpi[1][1] = max { Σ | p_{1j}^{-1}| : nh+1 < j < nk}

rnpi[2][1] = max { Σ | p_{2j}^{-1}| : nh+1 < j < nk}

rnpi[3][1] = max { Σ | p_{ij}^{-1}| : nh+1 < j < nk}

where the summations are taken over i = 3, ..., nk. The bound for $\|P_{i,j}^{-1}\|$ is given by rnpi[i][j]. These factors multiplied by the norms contained in errp give the contribution for $P^{-1}.E.P.$

(g) Check that the Newton map is a contraction on the ball.

The program checks that dN, the derivative of the approximate Newton map, has norm (with respect to the p-basis) less than 1. It also checks that N maps the ball about the approximate fixed point to itself. In particular it verifies that the hypotheses of Proposition 3.3 are satisfied. If the map is not a contraction then an error message is printed. Otherwise a message is printed that N is a contraction map on the ball around the approximate fixed point and there is a unique fixed point (h_*, k_*) in this ball.

(h) Estimate β_* and check that $g(\beta_*^6)^3 = \beta_*^6$, and $Dg(\beta_*^6) < 0$ where $g = \beta_*^{-1}$.

It is a simple matter to print out bounds for β_*, from information about the fixed point. From equation A4.26, $\kappa(\kappa(\beta_*^2)) = \beta_*^2$, where $\kappa = \beta_*^{-1}.n_*$ so we know that $g(g(\beta_*^6)3)3 = \beta_*^6$, where $g = \beta_*^{-1}k_*$. The program shows that the function $g(x)^3$ has derivative less than -1 on an interval containing β_*^6 and $g(\beta_*^6)^3$. If $g(\beta_*^6) \neq \beta_*^6$, the mean value theorem implies that g^3 must have a point of derivative -1 in between β_*^6 and $g(\beta_*^6)^3$. Hence program shows that β_*^6 must be a fixed point of g^3. The following variables are calculated

i_bet6 = β^6, i_g = g = β^{-1}k, i_gb = g(β^6)

i_gb3 = $g(\beta^6)^3$, i_comb - interval containing β^6 and $g(\beta^6)^3$

i_dg - interval containing {Dg(x) : x ϵ i_comb}

i_dg3 - interval containing {D(g^3)(x) : x ϵ i_comb}

(i) Check that h, k have non zero derivative at 0.

This is completely straightforward. We find bounds on $Dh_*(0)$, $Dk_*(0)$ and show that these quantities are positive. The variables are

i_dh0 = Dh(0), i_dk0 = Dk(0).

(j) Check that the spectrum of dT_* consists of three simple eigenvalues λ_0, λ_1, λ_2 with the rest of the spectrum contained completely within the disc D(0, 0.875). This is done using the following method of Eckmann et al (1982).

<u>Proposition A7.1</u> Let B =

$$\begin{bmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \beta_0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \beta_1 \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \beta_2 \\ \gamma_0 & \gamma_1 & \gamma_2 & \delta \end{bmatrix} \quad \epsilon \quad \Lambda(\mathbb{R}^3 \oplus \ell_1, \ \mathbb{R}^3 \oplus \ell_1)$$

be a matrix of operators and let it be represented by B

$$\begin{bmatrix} A_{0,0} & A_{0,1} & B_{0,2} & B_0 \\ A_{1,0} & A_{1,1} & A_{1,2} & B_1 \\ A_{2,0} & A_{2,1} & A_{2,2} & B_2 \\ C_0 & C_1 & C_2 & D \end{bmatrix}$$

and we assume that <u>all</u> off-diagonal elements of this interval matrix are symmetric matrices. Then if the interval arithmetic determinant of this matrix does not contain 0, then the matrix B is invertible.

<u>proof</u> Let $u =$

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ v \end{bmatrix} \in \mathbb{R}^3 \oplus \ell_1$$

Then if $B(u) = 0$ we have for $i = 0, 1, 2$

$$\alpha_{i,0}.u_0 + \alpha_{i,1}.u_1 + \alpha_{i,2}.u_2 + \beta_i(v) = 0$$

and there is a $y_i \in B_i$ such that

$$\alpha_{i,0}.u_0 + \alpha_{i,1}.u_1 + \alpha_{i,2}.u_2 + y_i.\| v \| = 0$$

Similarly, there are $z_j \in C_j$, $w \in D$ such that

$$z_0.u_0 + z_1.u_1 + z_2.u_2 + w.\| v \| = 0$$

Thus $B(u) = 0$ implies that $0 \in B(U)$ where $U \neq 0$ is

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \| v \| \end{bmatrix} \in \mathbb{R}^4$$

However if $0 \notin \det(B)$, then $C(U) \neq 0$ for any matrix $C \in B$. This is a contradiction and the result is proved.

□

The reason for stipulating that even the off-diagonal elements $A_{i,j}$ are symmetric intervals is the following. Let Γ_1, Γ_2, Γ_3, Γ_4 disjoint be circles centred at points c_1, c_2, c_3, c_4 on the real axis.

<u>Proposition A7.2</u> Let B and B be as in Proposition A7.1. Then if $\det(B - \lambda I)$ does not contain 0 on the real axis outside of the circles Γ_i and if $\det(B - \lambda I)$ changes sign according to

$$(c_1 - \lambda).(c_2 - \lambda).(c_3 - \lambda).(c_4 - \lambda) \qquad \text{(A7.3)}$$

then B has no eigenvalues on the circles Γ_i.

<u>proof</u> If $\det(B - \lambda I)$ changes sign according to the sign of (A7.3) then this says that in the determinant the term (A7.3) in the determinant is greater in absolute value is greater than the sum of

the absolute value of the other terms in the determinant. However, since the off-diagonal elements of B are symmetric intervals, this says that the absolute value of (A7.3) is greater than the absolute value of the other terms for any λ in $\mathbb{C}$ not strictly contained within any of the circles Γ_i. □

The program first of all calculates rdtm4, the norm of the matrix i_dtm4. This provides an upper bound on the eigenvalues of dT. It is therefore unnecessary to consider values of λ outside of the interval [-rdtm4, rdtm4]. The program checks that

$$\text{rdtm4} \quad < 3.0$$

and then for $\lambda \in$ [-3.0, rdtm4] and outside of the circles Γ_1, ..., Γ_4 checks that λ is not an eigenvalue. This is done by checking that det(i_dtml4 - λ.I) does not contain 0. Furthermore, we check that the determinant changes sign according to the sign of the product

$$(\lambda + 2\ 1441151880758587/2^{54}).(\lambda + 1).\lambda.(\lambda - 2\ 1261007895663739/2^{53})$$

Then from Propositions A7.1 and A7.2 we may conclude that for all pairs of functions (h, k) in the neighbourhood of (h_0, k_0) we have that dT(h, k) contains no eigenvalue on the circles Γ_i, i = 1, 2, 3, 4. For each λ, the determinant is calculated using the routine i_det4. Note that the determinant is evaluated for <u>all</u> the λ in i_lam. If the sign of the determinant is not as expected, an error message is printed. If all goes well, a message confirming that the determinant changes sign as expected is printed. The variables used are as follows:

gam - array containing the centres and radii of the circles Γ_i. gam[i][0], gam[i][1] - centre and radius of Γ_{i+1}.

i_dtml4 = i_dtm4 with off-diagonal elements replaced by symmetric intervals

rlammin, rlammax - bounds of the interval for which the

determinant is expected to have the same sign.

rlam0, rlam1 lower and upper bounds of the interval

i_lam of values of λ

sign expected sign of the determinant for rlam0 $<$ λ $<$ rlam1

i_d = i_det4(i_dtm14 - λ) - interval containing the values of the determinant.

rd1 = lower bound of i_d

rdu = upper bound of i_d

(k) Obtain good bounds for λ_0, λ_1, λ_2 and their eigenvectors and show that for λ_1 and λ_2 their eigenvectors violate the commuting conditions $dF_*^{(0)}(\delta h, \delta k) = 0$, $dF_*^{(3)}(\delta h, \delta k) = 0$.

For the each of λ_0, λ_1, λ_2 in turn we obtain better bounds for the exact eigenvalue. We also obtain bounds on the eigenvectors of these eigenvalues. For λ_1 we check that the condition $dF_*^{(0)}(\delta h, \delta k) = 0$ is not satisfied and for λ_2 we check that condition $dF_*^{(3)}(\delta h, \delta k) = 0$ is not satisfied. This section is rather long, so we subdivide into the following subsections

(i) Start loop for each eigenvalue $\ell = 0, 1, 2$.

(ii) Read in approximate eigenvalue and eigenvector, form newton map and estimate the accuracy of the approximate eigenvalue/eigenvector.

(iii) Form a ball around the approximate eigenvalue/eigenvector and calculate the derivative of the approximate Newton map N_i and show that it is a contraction on the ball. Print out accurate bound on eigenvalue.

(iv) Show that for $\ell = 1$ the condition $dF_*^{(0)}(\delta h, \delta k) = 0$ is violated and for $\ell = 2$ the condition $dF_*^{(3)}(\delta h, \delta k) = 0$ is violated.

(i) Start loop for each eigenvalue ℓ = 0, 1, 2,. The program opens the file "eigvec" which contains the approximate eigenvalues and eigenvectors. For ℓ = 0, 1, 2, it performs the following (ii) - (iv).

(ii) Read in approximate eigenvalue and eigenvector, form newton map and estimate the accuracy of the approximate eigenvalue/eigenvector.

The approximate eigenvalue and eigenvector is read in.

rlam0 = approximate eigenvalue λ'

u0 = approximate eigenvector $u' = (\delta h, \delta k)$

i_delh = δh, i_delk = δk

The routine i_dteval is caled to calculate $dT_*(\delta h, \delta k) = (i\_dth, i\_dtk)$.

i_su0 = $(dT_* - \lambda I)(\delta h, \delta k)$ (without the contribution of the error functions of i_dth, i_dtk).

The symmetric interval i_su0[nh] contains the norm of the high order function of i_dth. The symmetric interval i_su0[nk] contains the norm of the high order function of i_dtk.

i_u0p - $(\delta h, \delta k)$ expressed with respect to the p-basis

i_u0p = r_pi × i_u0.

The approximate Newton map N_i is defined as

I - M.S

where S is the map (3.4) and where $M \approx dS^{-1}$. In fact, the computer calculates its own matrix M. The precise choice of M is not important. The matrix M is given as

$$\begin{bmatrix} M_0 & & & \\ & M_1 & & \\ & & M_2 & \\ & & & M_3 \end{bmatrix}$$

where $M_i \in$ i_m4[i][i] i = 0, ..., 2, and M_3 = k.I where k $\in$ i_m4[3][3].

i_su0p = i_su0 expressed with respect to the p-basis.

i_su0p4 = i_su0p contracted to a 4 vector.

i_su0p4 has the error functions included in.

i_difp4 = i_m4 × i_su0p4

r_difp4 = Norm of i_difp4 - upper bound for

$$\| N_i(\lambda', u') - (\lambda', u') \|$$

rad = radius of ball around approximate eigenvalue/vector

i_u0p4 - i_u0 contracted to a 4 interval vector

i_up4 - 4 vector containing ball around i_u0p

The derivative matrix i_ds4 = dS consists of the matrix i_dtm4 - i_lam with the ℓth column replaced by the vector -i_up4. This is valid for all vectors in i_up4. The matrix representing the derivative of the approximate Newton map N_i, I - M.dS, is i_dn4. r_dn is the maximum column norm of i_dn4. The program now checks that rcontr > rdn + rdifp4/rad is less than 1 so that Proposition 3.3 gives a unique eigenvalue/vector in the ball around (λ', u'). The bound for the eigenvalue λ is printed out.

(iv) Show that for $\ell = 1$ the condition $dF_*^{(0)}(\delta h, \delta k) = 0$ is violated and for $\ell = 2$ the condition $dF_*^{(3)}(\delta h, \delta k) = 0$ is violated. This section is a straightforward application of the equations (A6.11) and (A6.12). First of all, if $\ell = 0$, then we skip to the end of the loop. Secondly we use rnp1, rnp2 (calculated above) to calculate a ball around the approximate eigenvector (i_delh, i_delk) in the standard basis that contains the exact eigenvalue and eigenvector pair. The calculation is then straightforward. The variables used are

i_delcom is an interval variable that (eventually) contains bounds on the right hand sides of equations (A6.11) and A(6.12).

i_twop = 2, i_three = 3

i_dh = Dh(β)

i_dk0 = Dk(0)

i_bet4 = β^4

i_delk0 = εk(0)

```
#Makefile for circ.dir

#declarations
CFLAGS =  -O

circ_proof :: circ_proof.o circ_rig.o i_function.o i_routines.o r_up.o \
            r_routines.o  r_function.o rsp_routines.o print.o
        cc -O -o circ_proof  circ_proof.o circ_rig.o i_function.o i_routines.o \
        r_up.o r_routines.o  r_function.o rsp_routines.o print.o -lm

circ_prep :: circ_prep.o circ_nonrig.o r_function.o
        cc -O -o circ_prep circ_prep.o circ_nonrig.o r_function.o -lm

checkproof :
        lint circ_proof.c circ_rig.c i_function.c i_routines.c r_up.c \
             r_routines.c  r_function.c rsp_routines.c print.c
```

```
/*
file: circ_prep.c
*/

/*-------------------------------------------------------------*/
/*                                                             */
/*                circ_prep.c                                  */
/*                                                             */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*                                                             */
/*      Renormalisation of Maps of the Circle                  */
/*                                                             */
/*      Program to find conjugating matrix p to                */
/*      transform the derivative matrix to an                  */
/*      approximate Jordan Normal Form                         */
/*                                                             */
/*                                                             */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*              includes and defines                           */
/*-------------------------------------------------------------*/


#include <stdio.h>                      /* standard I-O Library */
#include <math.h>                       /* mathematics library  */
#include "r_function.h"                 /* non-rigorous         */
                                        /* function routines    */

/*-------------------------------------------------------------*/
/*                        global variables                     */
/*-------------------------------------------------------------*/

int n;                                  /* deg of of polynomial */
                                        /* part of functions    */
int nh, nk;


double dtm[164][164];
double p[164][164];
double pi[164][164];
```

```
/*-------------------------------------------------------------*/
/*                        start of program                     */
/*-------------------------------------------------------------*/

main()
{

/*-------------------------------------------------------------*/
/*                        declarations                         */
/*-------------------------------------------------------------*/

FILE *par;          /* parameter file       */
FILE *fix;          /* approx. fixed  pt.   */
FILE *fppi;         /* conjugating mat.     */
real function h,k,th,tk;
double a1,r1,a2,r2;
register i,j;

/*-------------------------------------------------------------*/
/*                print out heading                            */
/*-------------------------------------------------------------*/


printf("%s%s%s",
       "Renormalisation of Circle Maps: Program to Calculate\n",
       "the basis change matrix\n",
       "-----------------------------------------\n");

/*-------------------------------------------------------------*/
/*              read in parameters and initialise dependent    */
/*              parameters                                     */
/*-------------------------------------------------------------*/


par = fopen("parameters","r");          /* open parameter file  */
                                        /* for reading          */
fscanf(par,"%d",&n);                    /* read n - deg of      */
                                        /* polynomial part      */

printf("n, deg of polynomial part = %d\n",n);

nh = n + 1;
nk = 2*n + 3;

printf("nh = %d, nk = %d\n", nh,nk);

fix = fopen("fixpoint","r");

                                        /* read centres and     */
                                        /* radii of domains     */
fscanf(fix,"%lf %lf %lf %lf",&a1,&r1,&a2,&r2);
```

```
printf("centres and radii of domains:\n");
printf("domain of h: a1 = %e, r1 = %e\n",a1,r1);
printf("domain of k: a2 = %e, r2 = %e\n",a2,r2);

/*-------------------------------------------------------------*/
/*              read in approximate fixed point                */
/*-------------------------------------------------------------*/

h = r_fzero(a1,r1);
for(i = 0; i <= 40; i++)                /* read h0             */
   fscanf(fix,"%lf",&h.p[i]);

k = r_fzero(a2,r2);
for(i = 0; i <= 40; i++)                /* read k0             */
   fscanf(fix,"%lf",&k.p[i]);

/*-------------------------------------------------------------*/
/*                 calculate derivative                        */
/*-------------------------------------------------------------*/

                        /* renormalise to set up various       */
                        /* dependent variables                 */
r_t(h, k, &th, &tk);

                        /* calculate derivative                */
r_dt(h, k, dtm);

/*-------------------------------------------------------------*/
/*                 calculate p,pi                              */
/*-------------------------------------------------------------*/

eigit(dtm, p, pi, nk);

/*-------------------------------------------------------------*/
/*                 scale p, pi                                 */
/*-------------------------------------------------------------*/

for(i = 0; i < nk; i++)
{
   if(i == nh)
      continue;
   for(j = 0; j < nk; j++)
   {
      if(j == nh)
         continue;
      p[i][j]  = 2.0*p[i][j];
      pi[i][j] = 0.5*pi[i][j];
   }
}
```

```
/*-------------------------------------------------------------*/
/*                 write out results                           */
/*-------------------------------------------------------------*/

fppi = fopen("ppi", "w");

for(i = 0; i <= nk; i++)
   for(j = 0; j <= nk; j++)
      fprintf(fppi,"%1.16e\n", p[i][j]);

for(i = 0; i <= nk; i++)
   for(j = 0; j <= nk; j++)
      fprintf(fppi,"%1.16e\n", pi[i][j]);

/*-------------------------------------------------------------*/
/*                 end                                         */
/*-------------------------------------------------------------*/

}

/*-------------------------------------------------------------*/
/*                 definitions and global varibales            */
/*-------------------------------------------------------------*/

#define max(r,s) ( (r) >= (s) ? (r) : (s) )

double colnorm[164];

/*-------------------------------------------------------------*/
/*                 fnorm(a,n)                                  */
/*-------------------------------------------------------------*/

double fnorm(a,n)
double a[164][164];
int n;
{
double rmax;
int i,j;

                                        /* set colnorm to 0    */
for(j = 0; j <= n; j++)
   colnorm[j] = 0.0;
                                        /* calculate col norms */
for(i = 0; i <= n; i++)
   for(j = 0; j <= n; j++)
      colnorm[j] += fabs(a[i][j]);
                                        /* return maximum column */
                                        /* norm of the submatrix */
                                        /* D                   */
rmax = 0.0;
for(j = 3; j <= n; j++)
   rmax = max(rmax, colnorm[j]);
return(rmax);
}
```

```
/*-------------------------------------------------------------*/
/*                  eigit(a,t,ti,n)                            */
/*-------------------------------------------------------------*/


eigit(a,t,ti,n)
double a[164][164];
double t[164][164];
double ti[164][164];
int n;
{

/* variable declarations */
double ep  = 1.0e-10;
double eps = 1.0e-15;
double eps1 = 0.001;
double eps2 = 0.7;
int i,j,k,it,mark= 0,left= 1,right = 1;
int m;
double akm,amk,h,g,hj,yh,aik,aim,te,tee,tep,tem,d,aki,ami;
double c,e,cx,sx,cot2x,sig,cotx,cos2x,sin2x,den,tanhy;
double chy,shy,c1,c2,s1,s2,tki,tmi,tik,tim,det;
double rnorm;

for(i = 0; i <= n; i++)
   for(j = 0; j <= n; j++)
      t[i][j] = 0.0;

for(i = 0; i <= n; i++)
   for(j = 0; j <= n; j++)
      ti[i][j] = 0.0;

for(i = 0; i <= n; i++)
{
  t[i][i]  = 1.0;
  ti[i][i] = 1.0;
}

for(it = 1; it <= 50; it++)
{
  printf("iteration number %d \n", it);
  rnorm = fnorm(a,n);
  printf("norm of bottom right block = %e \n", rnorm);
                                /*  if D has small norm stop   */
  if(rnorm < eps2)
     break;
  if(mark)
  {
    printf("mark = true \n");
    return;
  }
  mark = 1;
  for(k = 0; k < n; k++)
  {

for(m = k + 1; m <= n; m++)
{
  akm = a[k][m];
  amk = a[m][k];
                                /* only perform rotation if   */
                                /* necessary                  */
    if((colnorm[m] >= epsl || colnorm[k]  >= epsl)
    &&
    (fabs(amk+akm) >= epsl || fabs(akm-amk) >= epsl
    && a[m][m] - a[k][k] >= epsl) )
    {
      h = 0.0;
      g = 0.0;
      hj = 0.0;
      yh = 0.0;
      for(i = 0; i <= n; i++)
      {
        aik = a[i][k];
        aim = a[i][m];
        te = aik*aik;
        tee = aim*aim;
        yh = yh+te-tee;
        if(i != k && i != m)
        {
          aki = a[k][i];
          ami = a[m][i];
          h = h+aki*ami-aik*aim;
          tep = te+ami*ami;
          tem = tee+aki*aki;
          g = g+tep+tem;
          hj = hj-tep+tem;
        }
      }
      h = h+h;
      d = a[k][k]-a[m][m];
      akm = a[k][m];
      amk = a[m][k];
      c = akm+amk;
      e = akm-amk;
      if(fabs(c) <= ep)
      {
        cx = 1.0;
        sx = 0.0;
      }
```

```
else
{
  if(c == 0.0)
    printf("c = 0\n");
  cot2x = d/c;
  if(cot2x >= 0.0)
    sig = 1.0;
  else
    sig = -1.0;
  cotx = cot2x+(sig*sqrt(1.0+cot2x*cot2x));
  sx = sig/sqrt(1.0+cotx*cotx);
  cx = sx*cotx;
}

if(yh < 0.0)
{
  tem = cx;
  cx = sx;
  sx = -tem;
}
cos2x = cx*cx-sx*sx;
sin2x = 2.0*sx*cx;
d = d*cos2x+c*sin2x;
h = h*cos2x-hj*sin2x;
den = g+2.0*(e*e+d*d);
if(den == 0.0)
  printf("den = 0, m = %d, k =  %d \n",m,k);
else
{
tanhy = (e*d-h/2.0)/den;
if(fabs(tanhy) <= eps)
{
  chy = 1.0;
  shy = 0.0;
}
else
{
  if(1-tanhy*tanhy == 0.0)
    printf("1-tanhy = 0 \n");
  chy = 1.0/sqrt(1.0-tanhy*tanhy);
  shy = chy*tanhy;
}
c1 = chy*cx-shy*sx;
c2 = chy*cx+shy*sx;
s1 = chy*sx+shy*sx;
s2 = -chy*sx+shy*cx;
det = c1*c2-s1*s2;
if(det == 0.0)
  printf("det = 0, m = %d, k = %d\n" ,m,k);
if(fabs(s1) > ep || fabs(s2) > ep)
{
  mark = 0;
```

```
for(i = 0; i <= n; i++)
{
  aki = a[k][i];
  ami = a[m][i];
  a[k][i] = c1*aki+s1*ami;
  a[m][i] = s2*aki+c2*ami;
  if(left)
  {
    tki = ti[k][i];
    tmi = ti[m][i];
    ti[k][i] = c1*tki+s1*tmi;
    ti[m][i] = s2*tki+c2*tmi;
  }
}


for(i = 0; i <= n; i++)
{
  aik = a[i][k];
  aim = a[i][m];
  a[i][k] = (c2*aik-s2*aim)/det;
  a[i][m] = (-s1*aik+c1*aim)/det;
  if(right)
  {
    tik = t[i][k];
    tim = t[i][m];
    t[i][k] = (c2*tik-s2*tim)/det;
    t[i][m] = (-s1*tik+c1*tim)/det;
  }
}
}
}
}
}
}
}
}
return;
}
```

```
/*
file: circ_nonrig.c
*/

/*-------------------------------------------------------------*/
/*                                                             */
/*                      circ_nonrig.c                          */
/*                                                             */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*            t, dt - non-rigorous routines                    */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*            includes                                         */
/*-------------------------------------------------------------*/

#include "r_function.h"                 /* non-rigorous        */

/*-------------------------------------------------------------*/
/*                      global variables                       */
/*-------------------------------------------------------------*/

extern int n;                           /* deg of of polynomial */
                                        /* part of functions    */

double bet,bet2,bet3,beti;
real_function bx1,bx2;
real_function hb,hb2,hb3,kb,khb;

/*-------------------------------------------------------------*/
/*                      r_t(h,k,th,tk)                         */
/*-------------------------------------------------------------*/

/*  non_rigorous renormalisation transformation                */

r_t(h,k,th,tk)
real_function h,k;
real_function *th,*tk;
{

bet = r_eval(k,0.0);
bet2 = bet*bet;
bet3 = bet2*bet;
beti = 1/bet;

bx1 = r_fzero(h.a,h.r);
bx1.p[0] = bet3*h.a;
bx1.p[1] = bet3*h.r;

bx2 = r_fzero(k.a,k.r);
bx2.p[0] = bet3*k.a;
bx2.p[1] = bet3*k.r;

kb = r_fcompl(k,bx1);

*th = r_fsmult(beti,kb);

hb  = r_fcompl(h,bx2);
hb2 = r_fmult(hb,hb);
hb3 = r_fmult(hb,hb2);
khb = r_fcomp(k,hb3);

*tk = r_fsmult(beti,khb);

/*-------------------------------------------------------------*/
/*                          end                                */
/*-------------------------------------------------------------*/

return;
}

/*-------------------------------------------------------------*/
/*                      global variables                       */
/*-------------------------------------------------------------*/

extern int nh, nk;

/*-------------------------------------------------------------*/
/*                      r_dt(h,k,dtm)                          */
/*-------------------------------------------------------------*/

/*      non_rigorous derivative calculation                    */

r_dt(h,k,dtm)
real_function h,k;
double dtm[164][164];
{

double rbx1,rbx2,rhb3;
real_function sbx1,sbx2,shb3,dkb,dkbx,delb1,dkhb,hdkhb,
   dhb,dhbx,dkdh,delb2,dth,dtk,delkb,dlkhb,x1,x2;
double s0,delb;
register i,j,m;

/*-------------------------------------------------------------*/
/*            set up auxillary variables                       */
/*-------------------------------------------------------------*/

x1 = r_fzero(h.a,h.r);
x1.p[0] = h.a;
x1.p[1] = h.r;
```

```
x2 = r_fzero(k.a,k.r);
x2.p[0] = k.a;
x2.p[1] = k.r;

s0 = -k.a/k.r;

/*-------------------------------------------------------------*/
/*              print out analyticity improvement factors      */
/*-------------------------------------------------------------*/

printf("r_dt:analyticity improvement factors:\n");

sbx1 = r_fscale(bx1,k.a,k.r);
rbx1 = r_fnorm(sbx1);
printf("r_dt:norm of sbx1 = %e\n",rbx1);

sbx2 = r_fscale(bx2,h.a,h.r);
rbx2 = r_fnorm(sbx2);
printf("r_dt:norm of sbx2 = %e\n",rbx2);

shb3 = r_fscale(hb3,k.a,k.r);
rhb3 = r_fnorm(shb3);
printf("r_dt:norm of shb3 = %e\n",rhb3);

/*-------------------------------------------------------------*/
/*              various auxillary calculations                 */
/*-------------------------------------------------------------*/

dkb  = r_fdcompl(k,bx1);
dkbx = r_fmult2(dkb,x1);

delb1 = r_fdiff(r_fsmult(3*bet,dkbx), r_fsmult(1/bet2,kb));

dkhb  = r_fdcomp(k,hb3);
hdkhb = r_fmult(dkhb,hb2);
dhb   = r_fdcompl(h,bx2);
dhbx  = r_fmult2(dhb,x2);
dkdh  = r_fsmult(9.0*bet, r_fmult(hdkhb,dhbx));

delb2 = r_fdiff(dkdh, r_fsmult(1/bet2,khb));

/*-------------------------------------------------------------*/
/*              calculate derivative matrix                    */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/*                columns 0, n                                 */
/*-------------------------------------------------------------*/

                                        /* set top left of dtm too 0  */
for(i = 0; i <= nh; i++)
   for(j = 0; j <= nh; j++)
      dtm[i][j] = 0.0;

dtk = r_fsmult(3/bet,hdkhb);

for(j = 0; j <= n; j++)                 /* column loop      */
{
   for(i = 0, m = nh + 1; i <= n; i++, m++)  /* row loop       */
      dtm[m][j] = dtk.p[i];
   dtk = r_fmult2(dtk,sbx2);
}

/*-------------------------------------------------------------*/
/*                columns nh + 1, nk - 1                       */
/*-------------------------------------------------------------*/

delkb = r_fzero(h.a,h.r);
delkb.p[0] = bet1;
dlkhb = r_fzero(k.a,k.r);
dlkhb.p[0] = bet1;
delb = 1.0;
for(j = nh + 1; j < nk; j++)            /* column loop    */
{
   dth = r_fsum(delkb,r_fsmult(delb,delb1));
   dtk = r_fsum(dlkhb,r_fsmult(delb,delb2));

   for(i = 0, m = nh + 1; i <= n; i++, m++)  /* row loop       */
   {
      dtm[i][j] = dth.p[i];
      dtm[m][j] = dtk.p[i];
   }

delb *= s0;
delkb = r_fmult2(delkb,sbx1);
dlkhb = r_fmult(dlkhb,shb3);
}

/*-------------------------------------------------------------*/
/*              set rows and cols nh, nk equal to 0            */
/*-------------------------------------------------------------*/

for(i = 0; i <= nk; i++)
{
   dtm[nh][i] = 0.0;
   dtm[nk][i] = 0.0;
   dtm[i][nh] = 0.0;
   dtm[i][nk] = 0.0;
}
```

```
/*-------------------------------------------------------------*/
/*                    end                                      */
/*-------------------------------------------------------------*/

return;
}
```

```
/*
file: r_function.h
*/

/*-------------------------------------------------------------*/
/*                    typedefs                                 */
/*-------------------------------------------------------------*/

typedef struct {

        double p[81];           /* polynomial part             */
        double a;               /* centre of domain            */
        double r;               /* radius of domain            */

} real_function;

/*-------------------------------------------------------------*/
/*      declarations                                           */
/*-------------------------------------------------------------*/

real_function r_fzero();
double        r_fnorm();
real_function r_fsmult();
real_function r_fscale();
real_function r_fsum();
real_function r_fdiff();
real_function r_fmult();
real_function r_fmult2();
double        r_eval();
real_function r_fcomp();
real_function r_fcomp1();
real_function r_fdcomp();
real_function r_fdcomp1();
double        r_fdkeval();
```

```
/*
file: r_function.c
*/

/*-------------------------------------------------------------*/
/*              includes and defines                           */
/*-------------------------------------------------------------*/

#include <math.h>
#include "r_function.h"

#define abs(x) ((x) > 0.0 ? (x) : -(x) )

/*-------------------------------------------------------------*/
/*              global variables                               */
/*-------------------------------------------------------------*/

extern int n;                   /* degree of polynomial part   */

/*-------------------------------------------------------------*/
/*              r_fzero(a,r)                                   */
/*-------------------------------------------------------------*/

real_function r_fzero(a,r)
double a,r;
{
real_function f;
register i;

for(i = 0; i <= n; i++)
   f.p[i] = 0.0;
f.a = a; f.r = r;
return(f);
}

/*-------------------------------------------------------------*/
/*              r_fnorm(f)                                     */
/*-------------------------------------------------------------*/

double r_fnorm(f)
real_function f;
{
double s ;
register i;

s = 0.0;
for(i = 0; i <= n; i++)
   s += abs(f.p[i]);
return(s);
}
```

```
/*-------------------------------------------------------------*/
/*              r_fsmult(a,f)                                  */
/*-------------------------------------------------------------*/

real_function r_fsmult(a,f)
double a;
real_function f;
{
real_function g;
register i;

for(i = 0; i <= n; i++ )
   g.p[i] = a*f.p[i];
g.a = f.a; g.r = f.r;
return(g);
}

/*-------------------------------------------------------------*/
/*              r_fscale(f,a,r)                                */
/*-------------------------------------------------------------*/

real_function r_fscale(f,a,r)
real_function f;
double a;
double r;
{
real_function g;
double ri;
register i;

ri = 1.0/r;
g.p[0] = (f.p[0] - a)*ri;
for(i = 1; i <= n; i++)
   g.p[i] = f.p[i]*ri;
g.a = f.a; g.r = f.r;
return(g);
}

/*-------------------------------------------------------------*/
/*              r_fsum(f,g)                                    */
/*-------------------------------------------------------------*/


real_function r_fsum(f,g)
real_function f,g;
{
real_function h;
register i;

for(i = 0; i <= n; i++)
   h.p[i] = f.p[i] + g.p[i];
h.a = f.a; h.r = f.r;
return(h);
}
```

```
/*-------------------------------------------------------------*/
/*              r_fdiff(f,g)                                   */
/*-------------------------------------------------------------*/

real_function r_fdiff(f,g)
real_function f,g;
{
real_function h;
register i;

for(i = 0; i <= n; i++)
   h.p[i] = f.p[i] - g.p[i];
h.a = f.a; h.r = f.r;
return(h);
}

/*-------------------------------------------------------------*/
/*              r_fmult(f,g)                                   */
/*-------------------------------------------------------------*/

real_function r_fmult(f,g)
real_function f,g;
{
real_function h;
register i,j,k;

h = r_fzero(f.a,f.r);
for(i = 0; i <= n; i++)
   for(j = 0, k = i; j <= i; j++, k--)
      h.p[i] +=  f.p[j]*g.p[k];
return(h);
}

/*-------------------------------------------------------------*/
/*              r_fmult2(f,g)                                  */
/*-------------------------------------------------------------*/

real_function r_fmult2(f,g)
real_function f,g;
{
real_function h;
register i;

for(i = 0; i <= n; i++)
   h.p[i] = f.p[i]*g.p[0];
for(i = 1; i <= n; i++)
   h.p[i] += f.p[i-1]*g.p[1];
h.a = f.a; h.r = f.r;
return(h);
}
```

```
/*-------------------------------------------------------------*/
/*              r_eval(f,a)                                    */
/*-------------------------------------------------------------*/

double r_eval(f,a)
real_function f;
double a;
{
double b,c;
double r;
register i;

b = (a - f.a)/f.r;
r = abs(b);
if(r >= 1.0)
   printf("error in r_eval: r = %e\n", r);
c = f.p[n];
for(i = n - 1; i >= 0; i--)
   c = b*c + f.p[i];
return(c);
}

/*-------------------------------------------------------------*/
/*              r_fcomp(f,g)                                   */
/*-------------------------------------------------------------*/

real_function r_fcomp(f,g)
real_function f,g;
{
real_function h,q;
register i;
double r;

q = r_fscale(g, f.a, f.r);
r = r_fnorm(q);
if(r >= 1.0)
   printf("error in r_fcomp: norm of q = %e\n",r);
h = r_fzero(g.a,g.r);
h.p[0] = f.p[n];
for(i = n - 1; i >= 0; i--)
{
   h = r_fmult(h, q);
   h.p[0] +=  f.p[i];
}
return(h);
}
```

```
/*-------------------------------------------------------------*/
/*                  r_fcompl(f,g)                              */
/*-------------------------------------------------------------*/

real_function r_fcompl(f,g)
real_function f,g;
{
real_function h,q;
register i;
double r;

q = r_fscale(g, f.a, f.r);
r = r_fnorm(q);
if(r >= 1.0)
   printf("error in r_fcompl: norm of q = %e\n",r);
h = r_fzero(g.a, g.r);
h.p[0] = f.p[n];
for(i = n - 1; i >= 0; i--)
{
   h = r_fmult2(h, q);
   h.p[0] +=  f.p[i];
}
return(h);
}

/*-------------------------------------------------------------*/
/*                  r_fdcomp(f,g)                              */
/*-------------------------------------------------------------*/

real_function r_fdcomp(f,g)
real_function f,g;
{
real_function h,df;
double t;
register i;

t = r_fnorm(r_fscale(g,f.a,f.r));
if(t >= 1.0)
   printf("error in r_fdcomp: t = %e\n", t);
for(i = 1; i <= n; i++)
   df.p[i-1] =  ((double) i)*f.p[i];
df.p[n] = 0.0;
df.a = f.a; df.r = f.r;
h = r_fcomp(df, g);
return(r_fscale(h,0.0,f.r));
}
```

```
/*-------------------------------------------------------------*/
/*                  r_fdcompl(f,g)                             */
/*-------------------------------------------------------------*/

real_function r_fdcompl(f,g)
real_function f,g;
{
real_function h,df;
double t;
register i;

t = r_fnorm(r_fscale(g, f.a, f.r));
if(t >= 1.0)
   printf("error in r_fdcompl: t = %e\n", t);
for(i = 1; i <= n; i++)
   df.p[i-1] =  ((double) i)*f.p[i];
df.p[n] = 0.0;
df.a = f.a; df.r = f.r;
h = r_fcompl(df, g);
return(r_fscale(h,0.0,f.r));
}

/*-------------------------------------------------------------*/
/*                  r_fdkeval(f,k,a)                           */
/*-------------------------------------------------------------*/

double r_fdkeval(f,k,a)
real_function f;
int k;
double a;
{
real_function df;
double b,t;
int fac[81];
register i;

t = abs((a - f.a)/f.r);
if(t >= 1.0)
   printf("error in r_fdkeval: t = %e\n", t);
fac[0] = 1;
for(i = 2; i <= k; i++)
   fac[0] *= i;
for(i = 1; i <= n + 1; i++)
   fac[i] = ((k+i)*fac[i-1])/i;
for(i = 0; i <= n - k ; i++)
   df.p[i] =  ((double) fac[i])*f.p[i+k];
for(i = n - k + 1; i <= n; i++)
   df.p[i] = 0.0;
df.a = f.a; df.r = f.r;
b = r_eval(df, a);
return(b/pow(f.r, (double) k));
}
```

```
Renormalisation of Circle Maps: Program to Calculate
the basis change matrix
-------------------------------------------------
n, deg of polynomial part = 80
nh = 81, nk = 163
centres and radii of domains:
domain of h: a1 = -2.420692e-01, r1 = 2. 50000e-01
domain of k: a2 = 5.178627e-01, r2 = 5.900000e-01
r_dt:analyticity improvement factors:
r_dt:norm of sbx1 = 9.196637e-01
r_dt:norm of sbx2 = 9.348640e-01
r_dt:norm of shb3 = 9.772216e-01
iteration number 1
norm of bottom right block = 1.175577e+01
iteration number 2
norm of bottom right block = 7.496285e+00
iteration number 3
norm of bottom right block = 8.328038e+00
iteration number 4
norm of bottom right block = 8.155444e+00
iteration number 5
norm of bottom right block = 6.027016e+00
iteration number 6
norm of bottom right block = 4.480346e+00
iteration number 7
norm of bottom right block = 3.598350e+00
iteration number 8
norm of bottom right block = 3.200200e+00
iteration number 9
norm of bottom right block = 2.555304e+00
iteration number 10
norm of bottom right block = 1.682309e+00
iteration number 11
norm of bottom right block = 1.313338e+00
iteration number 12
norm of bottom right block = 1.190733e+00
iteration number 13
norm of bottom right block = 1.102702e+00
iteration number 14
norm of bottom right block = 9.201603e-01
iteration number 15
norm of bottom right block = 7.570611e-01
iteration number 16
norm of bottom right block = 6.982198e-01
```

number of terms = 40
a1 = -0.2420692426
r1 = 0.295
a2 = 0.5178627070
r2 = 0.590
h0 =
8.0301625772400195e-01
2.5319109346073096e-01
-1.4399532334899058e-02
-2.1397914343231543e-03
2.0791034591701196e-04
1.7519362174973529e-05
-2.9286757737429952e-06
-1.1130602288355899e-07
4.0681426752496120e-08
2.1486494375680110e-10
-5.2062328835527338e-10
1.2331738579984472e-11
6.1841292107144052e-12
-3.4718769152624776e-13
-6.8196092876076482e-14
6.6216070209119079e-15
6.7056573763038409e-16
-1.1124076853320829e-16
-7.6771488989406897e-18
4.5485355006054428e-19
-4.8239995705403061e-19
-2.2381640672051375e-19
-7.2093319271288236e-20
-2.2972603146356534e-20
-6.7029730095360806e-21
-1.7304234043645302e-21
-3.9927197075007734e-22
-8.2571498228397102e-23
-1.5215395378375848e-23
-2.4827880360876711e-24
-3.5671888248078272e-25
-4.4777820101249467e-26
-4.8559837054010166e-27
-4.4841458514200096e-28
-3.4647340133427299e-29
-2.1926551108893532e-30
-1.1053840961333806e-31
-4.2670794621547979e-33
-1.1847129342452259e-34
-2.1074764183031736e-36
-1.8056811144953243e-38

k =
-4.4865254344691955e-08
8.9893535801693121e-01
-1.2226021217359975e-01
-1.3732126810786901e-01
4.7925830275379180e-02
1.8374231095472819e-02
-1.3483474035944947e-02
-1.4032695124368876e-03
3.2313799911878730e-03
-3.0316956925274225e-04
-6.6883726385599941e-04
1.9912450492615242e-04
1.1334063875983182e-04
-6.7752343079048118e-05
-1.2766048879012870e-05
1.8125086137847167e-05
-5.9888010050540183e-07
-4.0897685221442101e-06
9.4303832058683884e-07
7.6951643124573009e-07
-3.7186206716135779e-07
-1.0659962152798379e-07
1.0826350097828152e-07
3.6345711252504950e-09
-2.6176652634938466e-08
4.3263188501579011e-09
5.3118146479830459e-09
-2.0731258177817106e-09
-8.4311635179970296e-10
6.5706507428821166e-10
6.9033453212445878e-11
-1.6919634430387238e-10
1.7231002948623372e-11
3.6674989708253253e-11
-1.1368317552883335e-11
-6.4438650537222183e-12
3.9789652850338634e-12
7.3995058254138642e-13
-1.0926518180870475e-12
4.1285332567143929e-14
2.5169665186948081e-13

```
/*
file: circ_proof.c
*/

/*-------------------------------------------------------------*/
/*                                                             */
/*                        circ_proof.c                         */
/*                                                             */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*                                                             */
/*      renormalisation of maps of the circle                  */
/*                                                             */
/*      program to prove the existence of a fixed              */
/*      point of the renormalisation transformation            */
/*      for rotation number the golden mean                    */
/*                                                             */
/*                                                             */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*              includes and defines                           */
/*-------------------------------------------------------------*/

#include <stdio.h>                      /* standard I-O library */
#include "r_up.h"                       /* rigorous error       */
                                        /* bounding routines    */
#include "r_routines.h"                 /* rigorous bounding of */
                                        /* arithmetic operations*/
#include "i_routines.h"                 /* interval arithmetic  */
                                        /* routines             */
#include "r_function.h"                 /* non-rigorous         */
                                        /* function routines    */
#include "i_function.h"                 /* rigorous interval    */
                                        /* function routines    */

#define max(r,s) ( (r) >= (s) ? (r) : (s) )
#define min(r,s) ( (r) <= (s) ? (r) : (s) )

/*-------------------------------------------------------------*/
/*                        global variables                     */
/*-------------------------------------------------------------*/

                                /* FILE variables               */

FILE *par;                      /* parameters file              */
FILE *fix;                      /* fixed point file             */
FILE *fppi;                     /* conjugating matrix file      */
FILE *feig;                     /* eigenval/vec file            */
```

```
                                /* interval function variables */
interval_function i_h0, i_k0, i_h, i_k, i_th0, i_tk0,
                  i_g, i_delh, i_delk, i_dth, i_dtk;

                                /* real function variables      */
real_function h0, k0;

                                /* interval variables           */
extern interval i_bet, i_bet2, i_bet3, i_beti;
interval i_a1, i_r1, i_a2, i_r2;
interval i_p[164][164], i_api[164][164], i_papi[164][164],
        (*i_e)[164] = i_papi, i_pi[164][164];
interval i_e2, (*i_e3)[164] = i_papi;
interval i_dif0[164], i_difp[164];
interval (*i_dtm)[164]  = i_papi;
interval (*i_temp)[164] = i_api;
interval i_dtm4[4][4], i_dti4[4][4], i_j4[4], i_dnm4[4][4],
        i_dtm14[4][4];
interval i_lam,i_d;
interval i_bet4,i_bet6,i_gb,i_gb3,i_comb,i_dg,i_gcomb,
        i_gcomb2,i_dg3;
interval i_dh0,i_dk0;
interval i_m4[4][4], i_dsm4[4][4], i_u0p4[4], i_su0p4[4],
        i_difp4[4], i_up4[4];
interval i_rad, i_delcom, i_lam0;
interval i_dhb, i_two, i_three, i_delk0, i_u0[164], i_u0p[164],
        i_su0[164], i_su0p[164];

                                /* double variables             */
double a1, r1;
double a2, r2;
double (*r_p)[164]   = (double (*)[]) i_papi;
double (*r_api)[164] = (double (*)[]) i_pi;
double e1, rad, rnp1, rnp2, rpi1, rpi2, rnpi[4][2];
double err[2][164], errp[2][164], rerr, rerrp;
double rcolsum, rmax, rdifp, rdifp4, rdnm4, rdtm4;
double gam[4][4], rlammax, rlammin, rlam1, rlam2, rd1, rdu;
double u0[164], rcontr, rlam0;

                                /* int variables                */
int n;
int nh,nk;
int sign, spec;

/*-------------------------------------------------------------*/
/*                        start of program                     */
/*-------------------------------------------------------------*/

main()
{
register i,j,l,m;
```

```
/*-------------------------------------------------------------*/
/*                print out heading                            */
/*-------------------------------------------------------------*/

printf("%s%s%s%s%s%s",
       "Program to Prove the Existence of a Fixed Point of the \n",
       "-------------------------------------------------------- \n",
       "Renormalisation Transformation for Cubic Critical Maps \n",
       "-------------------------------------------------------- \n",
       "of the Circle wth Golden Mean Rotation Number.\n",
       "-------------------------------------------------------- \n");

/*-------------------------------------------------------------*/
/*              read in parameters and initialise dependent    */
/*              variables                                      */
/*-------------------------------------------------------------*/

par = fopen("parameters","r");

fscanf(par,"%d",&n);                        /* read n - deg of    */
                                            /* polynomial part    */

printf("n, deg of polynomial part = %d\n",n);

nh = n + 1;
nk = 2*n + 3;

printf("nh = %d, nk = %d\n", nh, nk);

/*-------------------------------------------------------------*/
/*              read in approximate fixed point                */
/*-------------------------------------------------------------*/

fix = fopen("fixpoint","r");

                                            /* read centres and   */
                                            /* radii of domains   */

fscanf(fix,"%f %f %f %f", &a1,&r1,&a2,&r2);

printf("centres and radii of domains:\n");
printf("exact values for domains:\n");
print("a1 = %q\nr1 = %q\n",a1,r1);
print("a2 = %q\nr2 = %q\n",a2,r2);
printf("approximate values for domains:\n");
printf("a1 = %e\nr1 = %e\n",a1,r1);
printf("a2 = %e\nr2 = %e\n",a2,r2);

i_a1 = i_intrl(a1);
i_r1 = i_intrl(r1);
i_a2 = i_intrl(a2);
i_r2 = i_intrl(r2);
```

```
h0 = r_fzero(a1,r1);
for(i = 0; i <= 40; i++)                    /* read h0            */
   fscanf(fix,"%lf",&h0.p[i]);

k0 = r_fzero(a2,r2);
for(i = 0; i <= 40; i++)                    /* read k0            */
   fscanf(fix,"%lf",&k0.p[i]);

i_h0 = i_fconv(h0);                         /* convert h0,k0      */
i_k0 = i_fconv(k0);


/*-------------------------------------------------------------*/
/*    read in the basis change matrix r_p and its approx.      */
/*    inverse r_api and find interval matrices i_p, i_pi       */
/*    containing r_p and its inverse                           */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

fppi = fopen("ppi","r");

for(i = 0; i <= nk; i++)
   for(j = 0; j <= nk; j++)
       fscanf(fppi,"%lf",&r_p[i][j]);

for(i = 0; i <= nk; i++)
   for(j = 0; j <= nk; j++)
       fscanf(fppi,"%lf",&r_api[i][j]);

for(i = 0; i <= nk; i++)
   for(j = 0; j <= nk; j++)
   {
      i_p[i][j]   = i_intrl(r_p[i][j]);
      i_api[i][j] = i_intrl(r_api[i][j]);
   }

for(i = 0; i <= nk; i++)
{
   i_p[i][nh]   = i_zero;
   i_api[i][nh] = i_zero;
   i_p[i][nk]   = i_zero;
   i_api[i][nk] = i_zero;
   i_p[nh][i]   = i_zero;
   i_api[nh][i] = i_zero;
   i_p[nk][i]   = i_zero;
   i_api[nk][i] = i_zero;
}
i_p[nh][nh]   = i_one;
i_api[nh][nh] = i_one;
i_p[nk][nk]   = i_one;
i_api[nk][nk] = i_one;
```

```
for(i = 0; i <= nk; i++)
   for(j = 0; j <= nk; j++)
      i_pi[i][j] = i_api[i][j];

i_matmult(i_p, i_api, i_papi, nk);

for(i = 0; i <= nk; i++)           /* Note: i_e points to i_papi */
   i_e[i][i] = i_diff(i_e[i][i], i_one);

el = r_matcol(i_e, 0, nk, 0, nk);
if(el < 1.0)
   printf("basis change matrix invertible\n");
else
   printf("ERROR: basis change matrix not invertible, el = %f\n",el);

i_e2 = i_rmult(r_prod_up(el, r_inv_up(r_diff_down(1.0,el))),i_unit);

for(i = 0; i <= nk; i++)
   for(j = 0; j <= nk; j++)
   {
      i_e3[i][j] = i_e2;
      if(i == j)
         i_e3[i][i] = i_sum(i_e2, i_one);
   }

i_matmult(i_api, i_e3, i_pi, nk);

for(i = 0; i <= nk; i++)
{
   i_pi[i][nh]  = i_zero;
   i_pi[i][nk]  = i_zero;
   i_pi[nh][i]  = i_zero;
   i_pi[nk][i]  = i_zero;
}
i_pi[nh][nh] = i_one;
i_pi[nk][nk] = i_one;

/*-------------------------------------------------------------*/
/*       estimate accuracy of approx. fixed point              */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

i_t(i_h0, i_k0, &i_th0, &i_tk0);   /* renormalise              */

for(i = 0, m = nh + 1; i <= n; i++, m++)
{
  i_dif0[i] = i_diff(i_th0.p[i], i_h0.p[i]);
  i_dif0[m] = i_diff(i_tk0.p[i], i_k0.p[i]);
}
i_dif0[nh] = i_intr(-i_th0.h, i_th0.h);
i_dif0[nk] = i_intr(-i_tk0.h, i_tk0.h);
```

```
                                        /* convert to new basis      */
i_matvec(i_pi, i_dif0, i_difp, nk);

                                        /* form newton map           */
i_difp[0] = i_prod(i_intrl(-1/3.234), i_difp[0]);
i_difp[1] = i_prod(i_intrl( 1/1.239), i_difp[1]);
i_difp[2] = i_prod(i_intrl(-1/2.000), i_difp[2]);

rdifp = 0.0;
for(i = 0; i <= nk; i++)
   rdifp = r_sum_up(rdifp, r_abs(i_difp[i]));

print("norm of N(h0, k0) - (h0, k0) bounded by %q,\n", rdifp);
print("which is approx. %e\n", rdifp);

/*-------------------------------------------------------------*/
/*   form a ball around the approximate fixed point            */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

i_h = i_h0;
i_k = i_k0;

rad = 10*rdifp;
print("radius of ball around (h0, k0) = %q,\n", rad);
print("which is approx. %e\n", rad);

rnp1 = r_matcol(i_p, 0, nk, 0, nh);
rnp2 = r_matcol(i_p, 0, nk, nh + 1, nk);

i_h.e = r_prod_up(rad, rnp1);
i_k.e = r_prod_up(rad, rnp2);

print("radius about h0 (standard basis) = %q,\n", i_h.e);
print("which is approx. %e\n", i_h.e);
print("radius about k0 (standard basis) = %q,\n", i_k.e);
print("which is approx. %e\n", i_k.e);

/*-------------------------------------------------------------*/
/*        calculation of the derivative matrix                 */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

/* first renormalise to set up constants used by i_dt */
i_t(i_h, i_k, &i_th0, &i_tk0);
i_dt(i_h, i_k, i_dtm, err);
```

```
/*-------------------------------------------------------------*/
/*        conjugate derivative matrix to the new basis         */
/*-------------------------------------------------------------*/

i_matmult(i_pi, i_dtm, i_temp, nk);
i_matmult(i_temp, i_p, i_dtm, nk);

/*-------------------------------------------------------------*/
/*       contract derivative to a 4 x 4 matrix                 */
/*-------------------------------------------------------------*/

rnpi[0][0] = r_matcol(i_pi, 0, nh, 0, 0);
rnpi[1][0] = r_matcol(i_pi, 0, nh, 1, 1);
rnpi[2][0] = r_matcol(i_pi, 0, nh, 2, 2);
rnpi[3][0] = r_matcol(i_pi, 0, nh, 3, nk);

rnpi[0][1] = r_matcol(i_pi, nh + 1, nk, 0, 0);
rnpi[1][1] = r_matcol(i_pi, nh + 1, nk, 1, 1);
rnpi[2][1] = r_matcol(i_pi, nh + 1, nk, 2, 2);
rnpi[3][1] = r_matcol(i_pi, nh + 1, nk, 3, nk);

for(i = 0; i <= 1; i++)
   for(j = 0; j <= nk; j++)
   {
      errp[i][j] = 0.0;
      for(m = 0; m <= nk; m++)
         errp[i][j] = r_sum_up(errp[i][j],
                      r_prod_up(err[i][m],r_abs(i_p[m][j]) ) );
   }


for(i = 0; i <= 2; i++)
   for(j = 0; j <= 2; j++)
   {
      rerr = r_sum_up(r_prod_up(rnpi[i][0], errp[0][j]),
             r_prod_up(rnpi[i][1], errp[1][j]) );
      i_dtm4[i][j] = i_sum(i_dtm[i][j], i_intr(-rerr, rerr) );
   }

for(i = 0; i <= 2; i++)
{
   rmax = 0.0;
   for(j = 3; j <= nk; j++)
   {
      rerr = r_sum_up(r_prod_up(rnpi[i][0], errp[0][j]),
             r_prod_up(rnpi[i][1], errp[1][j]) );
      rerr = r_sum_up(rerr, r_abs(i_dtm[i][j]) );
      rmax = max(rmax, rerr);
   }
   i_dtm4[i][3] =  i_intr(-rmax, rmax);
}
```

```
for(j = 0; j <= 2; j++)
{
   rcolsum = r_matcol(i_dtm, j, j, 3, nk);
   rerr = r_sum_up(r_prod_up(rnpi[3][0], errp[0][j]),
          r_prod_up(rnpi[3][1], errp[1][j]) );
   rerr = r_sum_up(rerr, rcolsum);
   i_dtm4[3][jj] = i_intr(-rerr, rerr);
}

rmax = 0.0;
for(j = 3; j <= nk; j++)
{
   rcolsum = r_matcol(i_dtm, j, j, 3, nk);
   rerr = r_sum_up(r_prod_up(rnpi[3][0], errp[0][j]),
          r_prod_up(rnpi[3][1], errp[1][j]) );
   rerr = r_sum_up(rerr, rcolsum);
   rmax = max(rmax, rerr);
}

i_dtm4[3][3] = i_intr(-rmax, rmax);

/*-------------------------------------------------------------*/
/*    write out contracted matrix                              */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

printf("i_dtm4: contracted derivative matrix(decimal approx):\n");
for(i = 0; i <= 3; i++)
   for(j = 0; j <= 3; j++)
      printf("i_dtm4[%d][%d] = [%23.16e, %23.16e]\n",i,j,
             i_dtm4[i][j].lo, i_dtm4[i][j].up);

/*-------------------------------------------------------------*/
/*          check that N is a contraction on                   */
/*          neighbourhood of (h0, k0)                          */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

/* form derivative of approx. Newton map N */
for(i = 0; i <= 3; i++)
   for(j = 0; j <= 3; j++)
      i_dti4[i][j] = i_dtm4[i][j];

for(i = 0; i <= 3; i++)
   i_dti4[i][i] = i_diff(i_dti4[i][i], i_one);

i_j4[0] = i_intrl(-1/3.234);
i_j4[1] = i_intrl( 1/1.239);
i_j4[2] = i_intrl(-1/2.000);
i_j4[3] = i_intrl(-1/1.000);
```

```
for(i = 0; i <= 3; i++)
   for(j = 0; j <= 3; j++)
      i_dnm4[i][j] = i_prod(i_j4[i], i_dti4[i][j]);

for(i = 0; i <= 3; i++)
   i_dnm4[i][i] = i_diff(i_one,i_dnm4[i][i]);

rdnm4 = r_matcol4(i_dnm4, 0, 3, 0, 3);

/* check that newton map is a contraction */
if(r_sum_up(rdnm4, r_prod_up(rdifp, r_inv_down(rad))) < 1.0)
{
   printf("N is a contraction on ball around (h0,k0)\n");
   printf("T has a unique fixed point in ball\n");
}
else
   printf("ERROR: Newton map is not a contraction\n");

/*-------------------------------------------------------------*/
/*          determinant  calculations                          */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

for(i = 0; i <= 2; i++)
   for(j = 0; j <= 2; j++)
      if(i != j)
         i_dtm14[i][j] = i_prod(i_dtm4[i][j], i_unit);

rdtm4 = r_matcol4(i_dtm4, 0, 3, 0, 3);

if(rdtm4 >= 3.0)
   printf("ERROR: rdtm4 = %e\n", rdtm4);

for(i = 0; i <= 3; i++)
{
   fscanf(par,"%lf %lf", &gam[i][0], &gam[i][1]);
   printf("circle %d:\n",i+1);
   print("centre = %q,\n",gam[i][0]);
   printf("which is approx. %e\n",gam[i][0]);
   print("radius = %q,\n",gam[i][1]);
   printf("which is approx. %e\n",gam[i][1]);
}

/*-------------------------------------------------------------*/
/*          check invertibilty of (dtm - lam) for various lam  */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

sign = 1;
spec  = 1;
```

```
for(l = 0; l <= 4; l++)
{
   if(l >= 1)
      rlammin = r_sum_down(gam[l-1][0],gam[l-1][1]);
   else
      rlammin = -3.0;
   if(l <= 3)
      rlammax = r_diff_up(gam[l][0],gam[l][1]);
   else
      rlammax = rdtm4;

   if(rlammin > rlammax)
   {
      printf("ERROR: rlammin > rlammax\n");
      continue;
   }

   rlaml = rlammin;
   while(rlaml <= rlammax)
   {
      rlam2 = rlaml + 0.05;
      i_lam = i_intr(rlaml, rlam2);
      for(i = 0; i <= 3; i++)
         i_dtm14[i][i] = i_diff(i_dtm4[i][i], i_lam);
      i_d = i_det4(i_dtm14);
      rdl = i_d.lo;
      rdu = i_d.up;
      if(sign*rdl <= 0.0 || sign*rdu <= 0.0)
      {
         printf("ERROR: sign of det wrong:\n");
         printf("rlaml = %e, rlam2 = %e, sign = %d\n",rlaml, rlam2, sign);
         printf("i_d = [%e, %e]\n",i_d.lo, i_d.up);
         spec = 0;
      }
      rlaml = rlam2;
   }
   sign = -sign;
}

if(spec)
   printf("determinant changes sign as expected\n");

/*-------------------------------------------------------------*/
/*                  estimate b = beta and show                 */
/*                  Dg(b^6) < 0, g(b^6)^3 = b^6, g = 1/b.k     */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

print("beta is contained in [%q, %q],\n",i_bet.lo,i_bet.up);
print("which is approx. [%23.16e, %23.16e]\n",i_bet.lo,i_bet.up);

printf("-----------------------------------\n");
```

```
i_bet6 = i_prod(i_bet3, i_bet3);
i_g    = i_fsmult(i_beti, i_k);
i_gb   = i_eval(i_g, i_bet6);
i_gb3 = i_prod(i_prod(i_gb, i_gb), i_gb);

i_comb.lo  =  min(i_bet6.lo, i_gb3.lo);
i_comb.up  =  max(i_bet6.up, i_gb3.up);
print("i_comb = [%q, %q]\n",i_comb.lo, i_comb.up);
print("which is approx. [%23.16e, %23.16e]\n",i_comb.lo,i_comb.up);

i_dg = i_fdkeval(i_g, 1, i_comb);
i_gcomb  = i_eval(i_g, i_comb);
i_gcomb2 = i_prod(i_gcomb, i_gcomb);
i_dg3    = i_prod(i_intil(3), i_prod(i_gcomb2, i_dg));
print("i_dg3 = [%q, %q]\n",i_dg3.lo, i_dg3.up);
print("which is approx. [%23.16e, %23.16e]\n",i_dg3.lo,i_dg3.up);

if(i_dg3.up >= -1.0 )
   printf("ERROR: Dg^3 not < -1\n");
else
{
   printf("g(b^6)^3 = b^6\n");
   printf("Dg(b^6) < 0\n");
}

/*-------------------------------------------------------------*/
/*              calculation of Dh(0), Dk(0)                    */
/*-------------------------------------------------------------*/

printf("-----------------------------------\n");

i_dh0 = i_fdkeval(i_h, 1, i_zero);
if(i_dh0.lo <= 0)
   printf("ERROR: h may not have positive derivative at 0\n");
else
   printf("h has positive derivative at 0\n");

i_dk0 = i_fdkeval(i_k, 1, i_zero);
if(i_dk0.lo <= 0)
   printf("ERROR: k may not have positive derivative at 0\n");
else
   printf("k has positive derivative at 0\n");

/*-------------------------------------------------------------*/
/*        obtaining bounds for eigenvalues and eigenvectors    */
/*        and checking that two eigenvectors violate the       */
/*        commuting conditions                                 */
/*-------------------------------------------------------------*/

i_two    = i_intil(2);
i_three  = i_intil(3);
rpi1 = r_matcol(i_pi, 0, nk, 0, nh);
rpi2 = r_matcol(i_pi, 0, nk, nh + 1, nk);
feig = fopen("eigvec", "r");
```

```
/*-------------------------------------------------------------*/
/*        start loop for top 3 eigenvalues                     */
/*-------------------------------------------------------------*/

for(l = 0; l <= 2; l++)
{
fscanf(feig, "%lf", &rlam0);
printf("-----------------------------------\n");
printf("eigenvalue no %d =approx %23.16e\n",l,rlam0);
i_lam0 = i_intrl(rlam0);
for(i = 0; i <= nk; i++)
   fscanf(feig, "%lf", &u0[i]);

for(i = 0; i <= nk; i++)
   i_u0[i] = i_intrl(u0[i]);

for(i = 0; i <= nk; i++)
{
   i_u0p[i] = i_zero;
   for(j = 0; j <= nk; j++)
      i_u0p[i] = i_sum(i_u0p[i], i_prod(i_pi[i][j], i_u0[j]) );
}

i_delh = i_fzero(i_h.a,i_h.r);
i_delk = i_fzero(i_k.a,i_k.r);

for(i = 0, m = nh + 1; i <= n; i++, m++)
{
   i_delh.p[i] = i_u0[i];
   i_delk.p[i] = i_u0[m];
}

i_dteval(i_delh, i_delk, &i_dth, &i_dtk);

for(i = 0, m = nh + 1; i <= n; i++, m++)
{
   i_su0[i] = i_diff(i_dth.p[i], i_rmult(rlam0, i_u0[i]));
   i_su0[m] = i_diff(i_dtk.p[i], i_rmult(rlam0, i_u0[m]));
}
i_su0[nh] = i_intr(-i_dth.h,i_dth.h);
i_su0[nk] = i_intr(-i_dtk.h,i_dtk.h);

i_matvec(i_pi, i_su0, i_su0p, nk);

/*-------------------------------------------------------------*/
/*        form matrix M                                        */
/*-------------------------------------------------------------*/

/* clear i_m4 */
for(i = 0; i <= 3; i++)
   for(j = 0; j <= 3; j++)
      i_m4[i][j] = i_zero;
```

```
for(i = 0; i <= 3; i++)
   if(i == 1)
      i_m4[i][i] = i_intrl(-1/i_u0p[i].lo);
   else if(i < 3)
      i_m4[i][i] = i_intrl(1.0/(r_av(i_dtm4[i][i]) - rlam0));
   else
      i_m4[i][i] = i_intrl(-1.0/rlam0);

for(i = 0; i <= 3; i++)
   if(int_zero(i_m4[i][i]))
      printf("ERROR: i_m4[%d][%d] contains 0\n",i,i);

/*-------------------------------------------------------------*/
/*       find out how good the approx. solution is             */
/*-------------------------------------------------------------*/

for(i = 0; i <= 2; i++)
      i_su0p4[i] = i_su0p[i];

i_su0p4[3] = i_zero;
for(i = 3; i <= nk; i++)
   i_su0p4[3] = i_sum(i_su0p4[3], i_abs(i_su0p[i]));

i_su0p4[3] = i_prod(i_unit, i_su0p4[3]);

rerrp = r_sum_up(r_prod_up(rpi1, i_dth.e),
                 r_prod_up(rpi2, i_dtk.e));

for(i = 0; i <= 3; i++)
   i_su0p4[i] = i_sum(i_su0p4[i], i_intr(-rerrp, rerrp) );

i_matvec4(i_m4, i_su0p4, i_difp4, 3);

rdifp4 = 0.0;
for(i = 0; i <= 3; i++)
   rdifp4 = r_sum_up(rdifp4, r_abs(i_difp4[i]));

prinntf("norm of Ni(lam',u') - (lam',u') = %e\n", rdifp4);

/*-------------------------------------------------------------*/
/*        form ball around (lam', u')                          */
/*-------------------------------------------------------------*/

for(i = 0; i <= 2; i++)
      i_u0p4[i] = i_u0p[i];

i_u0p4[3] = i_zero;
for(i = 3; i <= nk; i++)
   i_u0p4[3] = i_sum(i_u0p4[3], i_abs(i_u0p[i]));

i_u0p4[3] = i_prod(i_unit, i_u0p4[3]);
```

```
rad = 10.0*rdifp4;
i_rad = i_intr(-rad, rad);
i_lam = i_sum(i_lam0, i_rad);
for(i = 0; i <= 3; i++)
   if(i == 1)
      i_up4[i] = i_u0p4[i];
   else
      i_up4[i] = i_sum(i_u0p4[i],i_rad);


/*-------------------------------------------------------------*/
/*       form derivative matrix dS                             */
/*-------------------------------------------------------------*/

for(j = 0; j <= 3; j++)
   if(j == 1)
      for(i = 0; i <= 3; i++)
            i_dsm4[i][j] = i_neg(i_up4[i]);
   else
      for(i = 0; i <= 3; i++)
      {
         i_dsm4[i][j] = i_dtm4[i][j];
         if(i == j)
            i_dsm4[i][i] = i_diff(i_dsm4[i][i], i_lam);
      }

/*-------------------------------------------------------------*/
/*        show Ni is contraction on ball about (lam',u')       */
/*-------------------------------------------------------------*/

i_matmult4(i_m4, i_dsm4, i_dnm4, 3);

for(i = 0; i <= 3; i++)
   i_dnm4[i][i] = i_diff(i_one, i_dnm4[i][i]);

rdnm4 = r_matcol4(i_dnm4, 0, 3, 0, 3);

printf("norm of derivative = %e\n", rdnm4);

rcontr = r_sum_up(rdnm4, r_prod_up(rdifp, r_inv_up(rad)));
if(rcontr < 1.0)
   printf("Ni is a contraction on ball\n");
else
   printf("ERROR: Ni is not a contraction on ball\n");

print("eigenvalue %d contained in [%q, %q]\n",1,i_lam.lo,i_lam.up);
print("which is approx. [%23.16e, %23.16e]\n",i_lam.lo,i_lam.up);

/*-------------------------------------------------------------*/
/*       if 1 != 1 or 1 != 2   next 1                          */
/*-------------------------------------------------------------*/

if(1 == 0)
   continue;
```

```
/*-------------------------------------------------------------*/
/*       form ball about eigenvector in the old basis          */
/*-------------------------------------------------------------*/

i_delh = i_fzero(i_h.a, i_h.r);
i_delk = i_fzero(i_k.a, i_k.r);

for(i = 0, m = nh + 1; i <= n; i++, m++)
{
   i_delh.p[i] = i_u0[i];
   i_delk.p[i] = i_u0[m];
}

i_delh.e = r_prod_up(rnp1, rad);
i_delk.e = r_prod_up(rnp2, rad);

i_delcom = i_zero;

/*-------------------------------------------------------------*/
/*       if l == 1 check first commuting condition             */
/*-------------------------------------------------------------*/

if(l == 1)
{
   i_delcom = i_eval(i_delh, i_bet3);
   i_delcom = i_sum(i_delcom,
              i_prod(i_three, i_prod(i_fdkeval(i_h, 1, i_bet3),
              i_prod(i_bet2, i_eval(i_delk, i_zero) ) ) ) );
   i_delcom = i_diff(i_delcom, i_eval(i_delk, i_one));
   if(int_zero(i_delcom))
   {
      printf("ERROR: first condition not necessarily violated\n");
      printf("i_delcom = [%e, %e]\n",i_delcom.lo,i_delcom.up);
   }
   else
      printf("eigenvector number 1 violates dF*(0)(dh,dk) = 0\n");
}


/*-------------------------------------------------------------*/
/*       if l == 2 check second commuting condition            */
/*-------------------------------------------------------------*/

if(l == 2)
{
   /* auxillary calculations */
   i_dhb   = i_fdkeval(i_h, 1, i_bet3);
   i_dk0   = i_fdkeval(i_k, 1, i_zero);
   i_bet4  = i_prod(i_bet2, i_bet2);
   i_delk0 = i_eval(i_delk, i_zero);
   i_delcom = i_prod(i_fdkeval(i_delh, 1, i_bet3),
              i_prod(i_bet2, i_dk0) );
   i_delcom = i_sum(i_delcom,
              i_prod(i_three, i_prod(i_fdkeval(i_h, 2, i_bet3),
              i_prod(i_bet4, i_prod(i_dk0, i_delk0) ) ) ) );
   i_delcom = i_sum(i_delcom, i_prod(i_two,i_prod(i_dhb,
              i_prod(i_bet, i_prod(i_dk0, i_delk0) ) ) ) );
   i_delcom = i_sum(i_delcom, i_prod(i_dhb, i_prod(i_bet2,
              i_fdkeval(i_delk, 1, i_zero) ) ) );
   i_delcom = i_diff(i_delcom, i_prod(i_fdkeval(i_delk, 1, i_one),
              i_fdkeval(i_h, 1, i_zero) ) );
   i_delcom = i_diff(i_delcom, i_prod(i_fdkeval(i_k, 1, i_one),
              i_fdkeval(i_delh, 1, i_zero) ) );
   if(int_zero(i_delcom))
   {
      printf("ERROR: second condition not necessarily violated\n");
      printf("i_delcom = [%e, %e]\n",i_delcom.lo,i_delcom.up);
   }
   else
      printf("eigenvector number 2 violates dF*(3)(dh,dk) = 0\n");
}

/*-------------------------------------------------------------*/
/*       end eigenvalue loop 1                                 */
/*-------------------------------------------------------------*/

}
fclose(feig);

/*-------------------------------------------------------------*/
/*       end of program                                        */
/*-------------------------------------------------------------*/

}
```

```
/*
file: circ_rig.c
*/

/*-------------------------------------------------------------*/
/*                                                             */
/*                      circ_rig.c                             */
/*                                                             */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*             t, dt - rigorous routines                       */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*             includes                                        */
/*-------------------------------------------------------------*/

#include "r_up.h"                       /* rigorous error      */
                                        /* routines            */
#include "r_routines.h"                 /* rigorous bounding of */
                                        /* arithmetic operations*/
#include "i_routines.h"                 /* interval arithmetic */
                                        /* routines            */
#include "i_function.h"                 /* rigorous interval   */
                                        /* function routines   */

/*-------------------------------------------------------------*/
/*                      global variables                       */
/*-------------------------------------------------------------*/

extern int n;                           /* deg of of polynomial */
                                        /* part of functions    */

interval i_bet,i_bet2,i_bet3,i_beti;
interval_function i_bx1,i_bx2;
interval_function i_hb,i_hb2,i_hb3,i_kb,i_khb;

/*-------------------------------------------------------------*/
/*                    i_t(i_h,i_k,i_th,i_tk)                   */
/*-------------------------------------------------------------*/

/*  rigorous renormalisation transformation                    */

i_t(i_h,i_k,i_th,i_tk)
interval_function i_h,i_k,*i_th,*i_tk;
{

i_bet  = i_eval(i_k,i_zero);
i_bet2 = i_prod(i_bet,i_bet);
i_bet3 = i_prod(i_bet2,i_bet);
i_beti = i_inv(i_bet);

i_bx1 = i_fzero(i_h.a,i_h.r);
i_bx1.p[0] = i_prod(i_bet3,i_h.a);
i_bx1.p[1] = i_prod(i_bet3,i_h.r);

i_bx2 = i_fzero(i_k.a,i_k.r);
i_bx2.p[0] = i_prod(i_bet3,i_k.a);
i_bx2.p[1] = i_prod(i_bet3,i_k.r);

i_kb  = i_fcompl(i_k,i_bx1);

*i_th = i_fsmult(i_beti,i_kb);

i_hb  = i_fcompl(i_h,i_bx2);
i_hb2 = i_fmult(i_hb,i_hb);
i_hb3 = i_fmult(i_hb2,i_hb);
i_khb = i_fcomp(i_k,i_hb3);

*i_tk = i_fsmult(i_beti,i_khb);

/*-------------------------------------------------------------*/
/*           end                                               */
/*-------------------------------------------------------------*/

return;
}

/*-------------------------------------------------------------*/
/*                      global variables                       */
/*-------------------------------------------------------------*/

extern int nh,nk;
interval_function i_delb1,i_delb2,i_hdkhb;

/*-------------------------------------------------------------*/
/*                  i_dt(i_h, i_k, i_dtm, err)                 */
/*-------------------------------------------------------------*/

/*     rigorous derivative calculation                         */

i_dt(i_h,i_k,i_dtm,err)
interval_function i_h,i_k;
interval i_dtm [164][164];
double err[2][164];
{
interval_function i_sbx1,i_sbx2,i_shb3,i_dkb,i_dkbx,
   i_dkhb,i_dhb,i_dhbx,i_dkdh,i_dth,i_dtk,i_delh,i_delk,
   i_delhb,i_delkb,i_d1khb,i_x1,i_x2;
interval i_s0,i_delb,i_three;
double rbx1,rbx2,rhb3;
register i,j,m;
```

```
/*-------------------------------------------------------------*/
/*              set up auxillary variables                     */
/*-------------------------------------------------------------*/

i_xl = i_fzero(i_h.a,i_h.r);
i_xl.p[0] = i_h.a;
i_xl.p[1] = i_h.r;

i_x2 = i_fzero(i_k.a,i_k.r);
i_x2.p[0] = i_k.a;
i_x2.p[1] = i_k.r;

i_s0    = i_quot(i_neg(i_k.a),i_k.r);
i_three = i_intil(3);

/*-------------------------------------------------------------*/
/*              check that dt is compact                       */
/*-------------------------------------------------------------*/

printf("i_dt:analyticity improvement factors:\n");

i_sbxl = i_fscale(i_bxl,i_k.a,i_k.r);
rbxl   = r_ifnorm(i_sbxl);
printf("norm of sbxl = %e\n",rbxl);

i_sbx2 = i_fscale(i_bx2,i_h.a,i_h.r);
rbx2   = r_ifnorm(i_sbx2);
printf("norm of sbx2 = %e\n",rbx2);

i_shb3 = i_fscale(i_hb3,i_k.a,i_k.r);
rhb3   = r_ifnorm(i_shb3);
printf("norm of shb3 = %e\n",rhb3);

/* test to see whether dt is a compact operator */
if(rbxl < 1.0 && rbx2 < 1.0 && rhb3 < 1.0)
   printf("dt is a compact operator\n");
else
   printf("ERROR: dt is not a compact operator\n");

/*-------------------------------------------------------------*/
/*              various auxillary calculations                 */
/*-------------------------------------------------------------*/

i_dkb  = i_fdcompl(i_k,i_bxl);
i_dkbx = i_fmult2(i_dkb,i_xl);

i_delbl = i_fdiff(i_fsmult(i_prod(i_three,i_bet),i_dkbx),
          i_fsmult(i_inv(i_bet2),i_kb));
```

```
i_dkhb  = i_fdcomp(i_k,i_hb3);
i_hdkhb = i_fmult(i_dkhb,i_hb2);
i_dhb   = i_fdcompl(i_h,i_bx2);
i_dhbx  = i_fmult2(i_dhb,i_x2);
i_dkdh  = i_fsmult(i_prod(i_intil(9),i_bet),
          i_fmult(i_hdkhb,i_dhbx));

i_delb2 = i_fdiff(i_dkdh,i_fsmult(i_inv(i_bet2),i_khb));

/*-------------------------------------------------------------*/
/*              calculate derivative matrix                    */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*              columns 0, n                                   */
/*-------------------------------------------------------------*/

                                        /* set top left of i_dtm */
                                        /* and err to 0          */
for(i = 0; i <= nh; i++)
   for(j = 0; j <= nh; j++)
      i_dtm[i][j] = i_zero;

for(j = 0; j <= nh; j++)
   err[0][j] = 0.0;

                                        /* initial column        */
i_dtk = i_fsmult(i_prod(i_three,i_beti),i_hdkhb);
for(j = 0; j <= n; j++)                 /* column loop           */
{
   for(i = 0, m = nh + 1; i <= n; i++, m++)   /* row loop       */
      i_dtm[m][j] = i_dtk.p[i];
   i_dtm[nk][j] = i_intr(-i_dtk.h,i_dtk.h);
   err[1][j] = i_dtk.e;
   i_dtk = i_fmult2(i_dtk, i_sbx2);
}

/*-------------------------------------------------------------*/
/*              column nh                                      */
/*-------------------------------------------------------------*/

i_delh = i_fzero(i_h.a,i_h.r);
i_delh.h = 1.0;
i_delhb = i_fcompl(i_delh,i_bx2);
i_dtk = i_fsmult(i_prod(i_three,i_beti),i_fmult(i_hdkhb,i_delhb));

for(i = 0, m = nh + 1; i <= n ; i++, m++)
   i_dtm[m][nh] = i_dtk.p[i];
i_dtm[nk][nh] = i_intr(-i_dtk.h,i_dtk.h);
err[1][nh] = i_dtk.e;
```

```
/*-------------------------------------------------------------*/
/*              columns nh + 1, nk - 1                         */
/*-------------------------------------------------------------*/

i_delkb = i_fzero(i_h.a,i_h.r);
i_delkb.p[0] = i_beti;
i_dlkhb = i_fzero(i_k.a,i_k.r);
i_dlkhb.p[0] = i_beti;
i_delb = i_one;


for(j = nh + 1; j < nk; j++)                    /* column loop   */
{
   i_dth = i_fsum(i_delkb,i_fsmult(i_delb,i_delbl));
   i_dtk = i_fsum(i_dlkhb,i_fsmult(i_delb,i_delb2));

   for(i = 0, m = nh + 1; i <= n; i++, m++)     /* row loop      */
   {
      i_dtm[i][j] = i_dth.p[i];
      i_dtm[m][j] = i_dtk.p[i];
   }
   i_dtm[nh][j] = i_intr(-i_dth.h,i_dth.h);
   i_dtm[nk][j] = i_intr(-i_dtk.h,i_dtk.h);
   err[0][j] = i_dth.e;
   err[1][j] = i_dtk.e;

   i_delb  = i_prod(i_delb,i_s0);
   i_delkb = i_fmult2(i_delkb,i_sbx1);
   i_dlkhb = i_fmult(i_dlkhb,i_shb3);
}

/*-------------------------------------------------------------*/
/*              column nk                                      */
/*-------------------------------------------------------------*/

i_delk   = i_fzero(i_k.a,i_k.r);
i_delk.h = 1.0;
i_delb   = i_eval(i_delk,i_zero);
i_dth    = i_fsmult(i_delb,i_delbl);
i_dth    = i_fsum(i_dth,i_fsmult(i_beti,i_fcompl(i_delk,i_bx1)));
i_dlkhb  = i_fcomp(i_delk,i_hb3);
i_dtk    = i_fsum(i_fsmult(i_delb,i_delb2),i_fsmult(i_beti,i_dlkhb));
for(i = 0, m = nh + 1; i <= n; i++, m++)
{
   i_dtm[i][nk] = i_dth.p[i];
   i_dtm[m][nk] = i_dtk.p[i];
}
i_dtm[nh][nk] = i_intr(-i_dth.h,i_dth.h);
i_dtm[nk][nk] = i_intr(-i_dtk.h,i_dtk.h);
err[0][nk] = i_dth.e;
err[1][nk] = i_dtk.e;

/*-------------------------------------------------------------*/
/*              end                                            */
/*-------------------------------------------------------------*/

return;
}

/*-------------------------------------------------------------*/
/*              i_dteval                                       */
/*-------------------------------------------------------------*/

i_dteval(i_delh,i_delk,i_dth,i_dtk)
interval_function i_delh,i_delk,*i_dth,*i_dtk;
{
interval i_delb;

/* Note: This routine must be preceeded by a call to i_t and to */
/* i_dt in order to initialise certain variables                */

/*-------------------------------------------------------------*/
/*              delh terms                                     */
/*-------------------------------------------------------------*/

*i_dtk = i_fsmult(i_prod(i_intil(3),i_beti),
         i_fmult(i_hdkhb,i_fcompl(i_delh,i_bx2) ) );

/*-------------------------------------------------------------*/
/*              delk terms                                     */
/*-------------------------------------------------------------*/

i_delb   = i_eval(i_delk,i_zero);
*i_dth   = i_fsmult(i_delb,i_delbl);
*i_dth   = i_fsum(*i_dth,i_fsmult(i_beti,i_fcompl(i_delk,i_bx1)));
*i_dtk   = i_fsum(*i_dtk,i_fsmult(i_delb,i_delb2));
*i_dtk   = i_fsum(*i_dtk,i_fsmult(i_beti,i_fcomp(i_delk,i_hb3)));

/*-------------------------------------------------------------*/
/*              end                                            */
/*-------------------------------------------------------------*/

return;
}
```

```
/*
file: i_function.h
*/

/*-------------------------------------------------------------*/
/*              typedefs                                       */
/*-------------------------------------------------------------*/

typedef struct {

        interval p[81];         /* polynomial part             */
        double h;               /* high order function norm    */
        double e;               /* error function norm         */
        interval a;             /* centre of domain            */
        interval r;             /* radius of domain            */

} interval_function;

/*-------------------------------------------------------------*/
/*      declarations                                           */
/*-------------------------------------------------------------*/

interval_function i_fzero();
interval_function i_fconv();
interval          i_fnorm();
double            r_ifnorm();
interval_function i_fsmult();
interval_function i_fscale();
interval_function i_fsum();
interval_function i_fdiff();
interval_function i_fmult();
interval_function i_fmult2();
interval          i_eval();
interval_function i_fcomp();
interval_function i_fcompl();
interval_function i_fdcomp();
interval_function i_fdcompl();
interval          i_fdkeval();
```

```
/*
file: i_function.c
*/

/*-------------------------------------------------------------*/
/*              includes and defines                           */
/*-------------------------------------------------------------*/

#include "r_up.h"
#include "r_routines.h"
#include "i_routines.h"
#include "i_function.h"
#include "r_function.h"

/*-------------------------------------------------------------*/
/*              global variables                               */
/*-------------------------------------------------------------*/

extern int n;                   /* degree of polynomial part   */

/*-------------------------------------------------------------*/
/*              i_fzero(a,r)                                   */
/*-------------------------------------------------------------*/

interval_function i_fzero(a,r)
interval a, r;
{
interval_function f;
register i;

for(i = 0; i <= n; i++)
   f.p[i] = i_zero;
f.h = f.e = 0.0;
f.a = a; f.r = r;
return(f);
}

/*-------------------------------------------------------------*/
/*              i_fconv(f)                                     */
/*-------------------------------------------------------------*/

interval_function i_fconv(f)
real_function f;
{
interval_function g;
register i;

for(i = 0; i <= n; i++)
   g.p[i] =  i_intrl(f.p[i]);
g.h = 0.0; g.e = 0.0;
g.a = i_intrl(f.a); g.r = i_intrl(f.r);
return(g);
}
```

```
/*-------------------------------------------------------------*/
/*                  i_fnorm(f)                                 */
/*-------------------------------------------------------------*/

interval i_fnorm(f)
interval_function f;
{
interval s;
register i;

s = i_zero;
for(i = 0; i <= n; i++)
   s = i_sum(s,i_abs(f.p[i]));
s = i_sum(s,i_intr(0.0, f.h));
s = i_sum(s,i_intr(-f.e, f.e));
return(s);
}

/*-------------------------------------------------------------*/
/*                  r_ifnorm(f)                                */
/*-------------------------------------------------------------*/

double r_ifnorm(f)
interval_function f;
{
return(r_abs(i_fnorm(f)));
}

/*-------------------------------------------------------------*/
/*                  i_fsmult(a,f)                              */
/*-------------------------------------------------------------*/

interval_function i_fsmult(a,f)
interval a;
interval_function f;
{
interval_function g;
double s;
register i;

for(i = 0; i <= n; i++ )
   g.p[i] = i_prod(a,f.p[i]);
s = r_abs(a);
g.h = r_prod_up(s,f.h);
g.e = r_prod_up(s,f.e);
g.a = f.a; g.r = f.r;
return(g);
}
```

```
/*-------------------------------------------------------------*/
/*                  i_fscale(f,a,r)                            */
/*-------------------------------------------------------------*/

interval_function i_fscale(f,a,r)
interval_function f;
interval a;
interval r;
{
interval_function g;
interval ri;
double s;
register i;

ri = i_inv(r);
f.p[0] = i_diff(f.p[0],a);
for(i = 0; i <= n; i++)
   g.p[i] = i_prod(f.p[i],ri);
s = r_abs(ri);
g.h = r_prod_up(s,f.h);
g.e = r_prod_up(s,f.e);
g.a = f.a; g.r = f.r;
return(g);
}

/*-------------------------------------------------------------*/
/*                  i_fsum(f,g)                                */
/*-------------------------------------------------------------*/

interval_function i_fsum(f,g)
interval_function f,g;
{
interval_function h;
register i;

for(i = 0; i <= n; i++)
   h.p[i] = i_sum(f.p[i],g.p[i]);
h.h = r_sum_up(f.h, g.h);
h.e = r_sum_up(f.e, g.e);
h.a = f.a; h.r = f.r;
return(h);
}
```

```
/*-------------------------------------------------------------*/
/*                  i_fdiff(f,g)                               */
/*-------------------------------------------------------------*/

interval_function i_fdiff(f,g)
interval_function f,g;
{
interval_function h;
register i;

for(i = 0; i <= n; i++)
   h.p[i] = i_diff(f.p[i], g.p[i]);
h.h = r_sum_up(f.h, g.h);
h.e = r_sum_up(f.e, g.e);
h.a = f.a; h.r = f.r;
return(h);
}

/*-------------------------------------------------------------*/
/*                  i_fmult(f,g)                               */
/*-------------------------------------------------------------*/

interval_function i_fmult(f,g)
interval_function f,g;
{
interval_function h;
interval s;
double aa = 0.0, ab = 0.0, ac = 0.0;
register i,j,k;

h = i_fzero(f.a,f.r);
for(i = 0; i <= n; i++)
   for(j = 0, k = i; j <= i; j++, k--)
      h.p[i] = i_sum(h.p[i],i_prod(f.p[j],g.p[k]));
for(i = 2*n; i > n; i--)
{
   s = i_zero;
   for(j = i - n, k = n; j <= n; j++, k--)
      s = i_sum(s, i_prod(f.p[j], g.p[k]));
   aa = r_sum_up(aa, r_abs(s));
}

for(i = 0; i <= n; i++)
   ab =  r_sum_up(ab, r_abs(f.p[i]));
for(i = 0; i <= n; i++)
   ac =  r_sum_up(ac, r_abs(g.p[i]));
h.h = aa;
h.h = r_sum_up(h.h,r_sum_up(r_prod_up(ab,g.h),r_prod_up(f.h,ac)));
h.h = r_sum_up(h.h,r_sum_up(r_prod_up(f.e,g.h),r_prod_up(f.h,g.e)));
h.h = r_sum_up(h.h,r_prod_up(f.h,g.h));
h.e = r_prod_up(f.e,g.e);
h.e = r_sum_up(h.e,r_sum_up(r_prod_up(ab,g.e),r_prod_up(f.e,ac)));
return(h);
}
```

```
/*-------------------------------------------------------------*/
/*                  i_fmult2(f,g)                              */
/*-------------------------------------------------------------*/

interval_function i_fmult2(f,g)
interval_function f,g;
{
interval_function h;
double ac = 0.0;
register i;

h = i_fzero(f.a,f.r);
for(i = 0; i <= n; i++)
   h.p[i] = i_prod(f.p[i],g.p[0]);
for(i = 1; i <= n; i++)
   h.p[i] = i_sum(h.p[i],i_prod(f.p[i-1],g.p[1]));
ac = r_sum_up(r_abs(g.p[0]), r_abs(g.p[1]));
h.h = r_sum_up(r_prod_up(f.h,ac),r_prod_up(r_abs(f.p[n]),r_abs(g.p[1])));
h.e = r_prod_up(f.e,ac);
return(h);
}

/*-------------------------------------------------------------*/
/*                  i_eval(f,a)                                */
/*-------------------------------------------------------------*/

interval i_eval(f,a)
interval_function f;
interval a;
{
interval b,c;
double r,rh;
register i;

b = i_quot(i_diff(a,f.a), f.r);
r = r_abs(b);
if(r >= 1.0)
   printf("error in i_eval: r = %e\n", r);
c = f.p[n];
for(i = n - 1; i >= 0; i--)
   c = i_sum(i_prod(b,c),f.p[i]);
rh = r_prod_up(r_power_up(r,n+1),f.h);
c = i_sum(c,i_intr(-rh,rh) );
c = i_sum(c,i_intr(-f.e,f.e) );
return(c);
}
```

```
/*-------------------------------------------------------------*/
/*              i_fcomp(f,g)                                   */
/*-------------------------------------------------------------*/

interval_function i_fcomp(f,g)
interval_function f,g;
{
interval_function h,q;
interval s,sl,sn,sln;
double r;
register i;

q = i_fscale(g,f.a, f.r);
s = i_fnorm(q);
r = r_abs(s);
if(r >= 1.0)
   printf("error in i_fcomp: norm of q = %e\n",r);
h = i_fzero(g.a,g.r);
h.p[0] = f.p[n];
for(i = n - 1; i >= 0; i--)
{
   h = i_fmult(h,q);
   h.p[0] = i_sum(h.p[0],f.p[i]);
}
q.p[0] = i_zero;
sl  = i_fnorm(q);
sn  = i_power(s,n+1);
sln = i_power(sl,n+1);
h.h = r_sum_up(h.h,r_prod_up(f.h,r_abs(sln)));
h.e = r_sum_up(h.e,r_sum_up(f.e,r_prod_up(f.h,
      r_abs((i_diff(sn,sln))))));
return(h);
}
```

```
/*-------------------------------------------------------------*/
/*              i_fcompl(f,g)                                  */
/*-------------------------------------------------------------*/

interval_function i_fcompl(f,g)
interval_function f,g;
{
interval_function h,q;
interval s,sl,sn,sln;
double r;
register i;

q = i_fscale(g,f.a,f.r);
s = i_fnorm(q);
r = r_abs(s);
if(r >= 1.0)
   printf("error in i_fcompl: norm of q = %e\n",r);
h = i_fzero(g.a,g.r);
h.p[0] = f.p[n];
for(i = n - 1; i >= 0; i--)
{
   h = i_fmult2(h,q);
   h.p[0] = i_sum(h.p[0],f.p[i]);
}
q.p[0] = i_zero;
sl = i_fnorm(q);
sn  = i_power(s,n+1);
sln = i_power(sl,n+1);
h.h = r_sum_up(h.h,r_prod_up(f.h,r_abs(sln)));
h.e = r_sum_up(h.e,r_sum_up(f.e,r_prod_up(f.h,
      r_abs((i_diff(sn,sln))))));
return(h);
}
```

```
/*-------------------------------------------------------------*/
/*              i_fdcomp(f,g)                                  */
/*-------------------------------------------------------------*/

interval_function i_fdcomp(f,g)
interval_function f,g;
{
interval_function h,df;
double t,tdo,tup,u,v;
int m1,m2;
register i;

t = r_ifnorm(i_fscale(g,f.a,f.r));
if(t >= 1.0)
   printf("error in i_fdcomp: t = %e\n", t);
for(i = 1; i <= n; i++)
   df.p[i-1] =  i_prod(i_intil(i),f.p[i]);
df.p[n] = i_zero;
df.h = df.e = 0.0;
df.a = f.a; df.r = f.r;
h = i_fcomp(df,g);
tup = r_prod_up  (t, r_inv_up(r_diff_down(1.0, t)));
tdo = r_prod_down(t, r_inv_down(r_diff_up(1.0, t)));
m1 = (int) tdo;
m2 = m1 + 1;
if(m1 >= tdo || m2 <= tup)
   printf("error in i_fdcomp: m1 = %d, tdo = %e, tup = %e\n");
u = r_prod_up((double) m2, r_power_up(t, m1));
if(m1 < n)
   v = r_prod_up((double) (n + 1), r_power_up(t, n));
else
   v = u;
h.e = r_sum_up(h.e,r_prod_up(f.e,u));
h.e = r_sum_up(h.e,r_prod_up(f.h,v));
return(i_fscale(h,i_zero,f.r));
}
```

```
/*-------------------------------------------------------------*/
/*              i_fdcomp1(f,g)                                 */
/*-------------------------------------------------------------*/

interval_function i_fdcomp1(f,g)
interval_function f,g;
{
interval_function h,df;
double t,tdo,tup,u,v;
int m1,m2;
register i;

t = r_ifnorm(i_fscale(g,f.a,f.r));
if(t >= 1.0)
   printf("error in i_fdcomp1: t = %e\n", t);
for(i = 1; i <= n; i++)
   df.p[i-1] =  i_prod(i_intil(i) ,f.p[i]);
df.p[n] = i_zero;
df.h = df.e = 0.0;
df.a = f.a; df.r = f.r;
h = i_fcomp1(df,g);
tup = r_prod_up(t, r_inv_up(r_diff_down(1.0, t)));
tdo = r_prod_down(t, r_inv_down(r_diff_up(1.0, t)));
m1 = (int) tdo;
m2 = m1 + 1;
if(m1 >= tdo || m2 <= tup)
   printf("error in i_fdcomp1: m1 = %d, tdo = %e, tup = %e\n");
u = r_prod_up((double) m2, r_power_up(t, m1));
if(m1 < n)
   v = r_prod_up((double) (n + 1), r_power_up(t, n));
else
   v = u;
h.e = r_sum_up(h.e,r_prod_up(f.e,u));
h.e = r_sum_up(h.e,r_prod_up(f.h,v));
return(i_fscale(h,i_zero,f.r));
}
```

```
/*-------------------------------------------------------------*/
/*              i_fdkeval(f,k,a)                               */
/*-------------------------------------------------------------*/

interval i_fdkeval(f,k,a)
interval_function f;
int k;
interval a;
{
interval_function df;
interval b;
double t,tdo,tup,u,v,w;
register i;
int m1,m2,fac[81];

t = r_abs(i_quot(i_diff(a, f.a), f.r) );
if(t >= 1.0)
   printf("error in i_fdkeval: t = %e\n", t);
fac[0] = 1;
for(i = 2; i <= k; i++)
   fac[0] *= i;
for(i = 1; i <= n - k + 1; i++)
   fac[i] = ((k + i)*fac[i-1])/i;
for(i = 0; i <= n - k; i++)
   df.p[i] =  i_rmult((double) fac[i],f.p[i+k]);
for(i = n - k + 1;i <= n; i++)
   df.p[i] = i_zero;
df.e = df.h = 0.0;
df.a = f.a; df.r = f.r;
b = i_eval(df,a);
tup = r_prod_up((double) k, r_prod_up(t,
      r_inv_up(r_diff_down(1.0, t))));
tdo = r_prod_down((double) k, r_prod_down(t,
      r_inv_down(r_diff_up(1.0, t))));
m1  = (int) tdo;
m2  = m1 + 1;
if(m1 >=  tdo || m2 < tup)
   printf("error in i_fdkeval: m1 = %d, tdo = %e, tup = %e\n");
u = r_power_up(t,m1);
for(i = 1; i <= k; i++)
   u = r_prod_up(u,(double)(m1 + i));
if(m1 < n - k + 1)
   v = r_prod_up((double) fac[n-k+1], r_power_up(t,n-k+1));
else
   v = u;
w = r_prod_up(f.e, u);
w = r_sum_up(w, r_prod_up(f.h, v) );
b = i_sum(b,i_intr(-w, w) );
return(i_quot(b,i_power(f.r,k)));
}
```

```
/*
file: i_routines.h
*/

/*-------------------------------------------------------------*/
/*              typedefs                                       */
/*-------------------------------------------------------------*/

typedef struct  {
        double lo;      /* left  hand end point of interval     */
        double up;      /* right hand end point of interval     */
} interval;

/*-------------------------------------------------------------*/
/*              global variables                               */
/*-------------------------------------------------------------*/

extern interval i_zero;         /* zero interval [0,0]          */
extern interval i_one;          /* interval      [1,1]          */
extern interval i_unit;         /* unit interval [-1,1]         */

/*-------------------------------------------------------------*/
/*              declarations                                   */
/*-------------------------------------------------------------*/

interval i_intr();
interval i_intrl();
interval i_inti();
interval i_intil();
double   r_av();
interval i_neg();
interval i_sum();
interval i_diff();
interval i_rmult();
interval i_prod();
interval i_inv();
interval i_quot();
interval i_power();
interval i_abs();
double   r_abs();
int      int_zero();
void     i_matmult();
void     i_matvec();
double   r_matcol();
void     i_matmult4();
void     i_matvec4();
double   r_matcol4();
interval i_det4();
```

```
/*
file: i_routines.c
*/

/*-------------------------------------------------------------*/
/*              includes and defines                           */
/*-------------------------------------------------------------*/

#include "r_up.h"
#include "r_macros.h"
#include "i_routines.h"

#define max(r, s) ( (r) >= (s) ? (r) :  (s) )
#define min(r, s) ( (r) <= (s) ? (r) :  (s) )
#define abs(r)    ( (r) >= 0.0 ? (r) : -(r) )

/*-------------------------------------------------------------*/
/*              global variables                               */
/*-------------------------------------------------------------*/

interval i_zero = { 0.0, 0.0 } ;        /* zero interval [0,0]  */
interval i_one  = { 1.0, 1.0 } ;        /* interval [1,1]       */
interval i_unit = {-1.0, 1.0 } ;        /* unit interval [-1,1] */


/*-------------------------------------------------------------*/
/*              i_intr(r,s)                                    */
/*-------------------------------------------------------------*/

interval i_intr(r,s)    /* returns the interval [r,s], double   */
double r,s;             /* r,s                                  */
{
interval a;
a.lo = r;
a.up = s;
return(a);
}

/*-------------------------------------------------------------*/
/*              i_intrl(r)                                     */
/*-------------------------------------------------------------*/

interval i_intrl(r)     /* returns the interval [r,r], double   */
double r;               /* r                                    */
{
interval a;
a.lo = a.up = r;
return(a);
}
```

```
/*------------------------------------------------------------ */
/*              i_inti(i,j)                                    */
/*------------------------------------------------------------ */

interval i_inti(i,j)     /* returns the interval [i,j], int i,j */
int i,j;
{
interval a;
a.lo = (double) i;       /* convert i,j to double              */
a.up = (double) j;
return(a);
}

/*------------------------------------------------------------ */
/*              i_intil(i)                                     */
/*------------------------------------------------------------ */

interval i_intil(i)      /* returns the interval [i,i], int i   */
int i;
{
interval a;
a.lo = a.up = (double) i; /* convert i to double               */
return(a);
}

/*------------------------------------------------------------ */
/*              r_av(a)                                        */
/*------------------------------------------------------------ */

double r_av(a)           /* returns the middle point of the     */
interval a;              /* interval a                          */
{
return(0.5*(a.lo + a.up));
}

/*------------------------------------------------------------ */
/*              i_neg(a)                                       */
/*------------------------------------------------------------ */

interval i_neg(a)        /* returns the interval -a             */
interval a;
{
interval b;
b.lo = -a.up;
b.up = -a.lo;
return(b);
}
```

```
/*------------------------------------------------------------ */
/*              i_sum(a,b)                                     */
/*------------------------------------------------------------ */

interval i_sum(a,b)      /* returns an  interval c containing   */
interval a,b;            /* a + b                               */
{
interval c;
c.lo = r_sum_down(a.lo,b.lo);
c.up = r_sum_up  (a.up,b.up);
return(c);
}

/*------------------------------------------------------------ */
/*              i_diff(a,b)                                    */
/*------------------------------------------------------------ */

interval i_diff(a,b)     /* returns an interval c containing    */
interval a,b;            /* a - b                               */
{
interval c;
c.lo = r_diff_down(a.lo,b.up);
c.up = r_diff_up  (a.up,b.lo);
return(c);
}

/*------------------------------------------------------------ */
/*              i_rmult(r,a)                                   */
/*------------------------------------------------------------ */

interval i_rmult(r,a)    /* returns an interval c containing    */
double r;                /* r*a, r double, a interval           */
interval a;
{
interval b;
if(r == 0.0)
   b.lo = b.up = 0.0;
else if(r > 0.0)
{
   b.lo = r_prod_down(r,a.lo);
   b.up = r_prod_up  (r,a.up);
}
else
{
   b.lo = r_prod_down(r,a.up);
   b.up = r_prod_up  (r,a.lo);
}
return(b);
}
```

```
/*-------------------------------------------------------------- */
/*              i_prod(a,b)                                      */
/*-------------------------------------------------------------- */

interval i_prod(a,b)     /* returns an interval c containing     */
interval a,b;            /* a*b                                  */
{
interval c;
double r,s;              /* temporary storage variables          */

if(a.lo >= 0.0)
   if(b.lo >= 0.0)
   {
      c.lo = r_prod_down(a.lo, b.lo);
      c.up = r_prod_up  (a.up, b.up);
   }
   else if(b.up <= 0.0)
   {
      c.lo = r_prod_down(a.up, b.lo);
      c.up = r_prod_up  (a.lo, b.up);
   }
   else
   {
      c.lo = r_prod_down(a.up, b.lo);
      c.up = r_prod_up  (a.up, b.up);
   }
else if(a.up <= 0.0)
   if(b.lo >= 0.0)
   {
      c.lo = r_prod_down(a.lo, b.up);
      c.up = r_prod_up  (a.up, b.lo);
   }
   else if(b.up <= 0.0)
   {
      c.lo = r_prod_down(a.up, b.up);
      c.up = r_prod_up  (a.lo, b.lo);
   }
   else
   {
      c.lo = r_prod_down(a.lo, b.up);
      c.up = r_prod_up  (a.lo, b.lo);
   }
else
   if(b.lo >= 0.0)
   {
      c.lo = r_prod_down(a.lo, b.up);
      c.up = r_prod_up  (a.up, b.up);
   }
   else if(b.up <= 0.0)
   {
      c.lo = r_prod_down(a.up, b.lo);
      c.up = r_prod_up  (a.lo, b.lo);
   }
   else
   {
      r = r_prod_down(a.lo, b.up);
      s = r_prod_down(a.up, b.lo);
      c.lo = min(r, s);
      r = r_prod_up  (a.lo, b.lo);
      s = r_prod_up  (a.up, b.up);
      c.up = max(r, s);
   }

return(c);
}

/*-------------------------------------------------------------- */
/*              i_inv(a)                                         */
/*-------------------------------------------------------------- */

interval i_inv(a)                /* returns interval b          */
interval a;                      /* containing  1/a, a interval */
{                                /* 0 not in a                  */
interval b;
if(a.lo <= 0.0 && a.up >= 0.0)
   printf("error in i_inv: a = [%e, %e]\n",a.lo,a.up);
else
{
   b.lo = r_inv_down(a.up);
   b.up = r_inv_up  (a.lo);
}
return(b);
}



/*-------------------------------------------------------------- */
/*              i_quot(a,b)                                      */
/*-------------------------------------------------------------- */

interval i_quot(a,b)             /* returns an interval         */
interval a,b;                    /* containing a/b, a,b interval */
{                                /* 0 not in b                  */
return(i_prod(a,i_inv(b)));
}
```

```
/*-------------------------------------------------------------- */
/*              i_power(a,k)                                     */
/*-------------------------------------------------------------- */

interval i_power(a,k)    /* returns an interval b containing     */
interval a;              /* a**k for interval a >= 0 and k >= 0  */
int k;
{
interval b;
if(a.lo < 0.0 || k < 0)
   printf("error in i_power: a.lo = %e, k = %d\n",a.lo,k);
else
{
   b.lo = r_power_down(a.lo,k);
   b.up = r_power_up  (a.up,k);
}
return(b);
}

/*-------------------------------------------------------------- */
/*              i_abs(a)                                         */
/*-------------------------------------------------------------- */

interval i_abs(a)        /* returns an interval b containing     */
interval a;              /* the set of absolute values of        */
{                        /* elements of a                        */
interval b;
if(a.lo >= 0.0)
   return(a);
else if(a.up <= 0.0)
{
   b.lo = -a.up;
   b.up = -a.lo;
}
else
{
   b.lo = 0.0;
   b.up = max(a.up, -a.lo);
}
return(b);
}

/*-------------------------------------------------------------- */
/*              r_abs(a)                                         */
/*-------------------------------------------------------------- */

double r_abs(a)          /* returns an upper bound on the        */
interval a;              /* absolute value of elements of a      */
{
double r;                          /* temporary storage          */
r = -a.lo;
return(max(r, a.up));
}
```

```
/*-------------------------------------------------------------- */
/*              int_zero(a)                                      */
/*-------------------------------------------------------------- */

int int_zero(a)                  /* returns 1 (true) if 0 in a   */
interval a;                      /* else 0 (false)               */
{
if(a.lo <= 0.0 && a.up >= 0.0)
   return(1);
else
   return(0);
}

/*-------------------------------------------------------------*/
/*              i_matmult(a,b,c,m)                             */
/*-------------------------------------------------------------*/

void i_matmult(a,b,c,m)          /* returns in c the interval   */
interval a[164][164];            /* product of matrices a, b    */
interval b[164][164];            /* rows and cols range from 0  */
interval c[164][164];            /* to m                        */
int m;
{
interval col[164], d;
register i, j, k;

for(j = 0; j <= m; j++)
{
   for(i = 0; i <= m; i++)
      col[i] = b[i][j];
   for(i = 0; i <= m; i++)
   {
      d = i_zero;
      for(k = 0; k <= m; k++)
         d = i_sum(d, i_prod(a[i][k], col[k]) );
      c[i][j] = d;
   }
}
return;
}
```

```
/*-------------------------------------------------------------*/
/*            i_matvec(a,b,c,m)                                */
/*-------------------------------------------------------------*/

void i_matvec(a,b,c,m)          /* returns in c the interval   */
interval a[164][164];           /* product of matrix a with    */
interval b[164];                /* vector b                    */
interval c[164];                /* rows and cols range from    */
int m;                          /* 0 to m                      */
{
interval d;
register i, j;

for(i = 0; i <= m; i++)
{
   d = i_zero;
   for(j = 0; j <= m; j++)
      d = i_sum(d, i_prod(a[i][j], b[j]) );
   c[i] = d;
}
return;
}

/*-------------------------------------------------------------*/
/*                r_matcol(a,m1,m2,m3,m4)                      */
/*-------------------------------------------------------------*/

double r_matcol(a,m1,m2,m3,m4)  /* returns the max col sum norm */
interval a[164][164];           /* of the interval submatrix    */
int m1, m2, m3, m4;             /* of a, m1 <= j <= m2,         */
{                               /* m3 <= i <= m4                */
double rmax, rcolsum;
register i, j;

rmax = 0.0;
for(j = m1; j <= m2; j++)
{
   rcolsum = 0.0;
   for(i = m3; i <= m4; i++)
      rcolsum = r_sum_up(rcolsum, r_abs(a[i][j]));
   rmax = max(rmax, rcolsum);
}
return(rmax);
}
```

```
/*-------------------------------------------------------------*/
/*                i_matmult4(a,b,c,m)                          */
/*-------------------------------------------------------------*/

void i_matmult4(a,b,c,m)        /* returns in c the interval   */
interval a[4][4];               /* product of matrices a, b    */
interval b[4][4];               /* rows and cols range from 0  */
interval c[4][4];               /* to m                        */
int m;
{
interval col[4], d;
register i, j, k;

for(j = 0; j <= m; j++)
{
   for(i = 0; i <= m; i++)
      col[i] = b[i][j];
   for(i = 0; i <= m; i++)
   {
      d = i_zero;
      for(k = 0; k <= m; k++)
         d = i_sum(d, i_prod(a[i][k], col[k]) );
      c[i][j] = d;
   }
}
return;
}


/*-------------------------------------------------------------*/
/*            i_matvec4(a,b,c,m)                               */
/*-------------------------------------------------------------*/

void i_matvec4(a,b,c,m)         /* returns in c the interval   */
interval a[4][4];               /* product of matrix a with    */
interval b[4];                  /* vector b                    */
interval c[4];                  /* rows and cols range from    */
int m;                          /* 0 to m                      */
{
interval d;
register i, j;

for(i = 0; i <= m; i++)
{
   d = i_zero;
   for(j = 0; j <= m; j++)
      d = i_sum(d, i_prod(a[i][j], b[j]) );
   c[i] = d;
}
return;
}
```

```
/*-------------------------------------------------------------*/
/*                r_matcol4(a,m1,m2,m3,m4)                     */
/*-------------------------------------------------------------*/

double r_matcol4(a,m1,m2,m3,m4) /* returns the max col sum norm */
interval a[4][4];               /* of the interval submatrix    */
int m1, m2, m3, m4;             /* of a, m1 <= j <= m2,         */
                                /* m3 <= i <= m4                */
{
double rmax, rcolsum;
register i, j;

rmax = 0.0;
for(j = m1; j <= m2; j++)
{
   rcolsum = 0.0;
   for(i = m3; i <= m4; i++)
      rcolsum = r_sum_up(rcolsum, r_abs(a[i][j]));
   rmax = max(rmax, rcolsum);
}
return(rmax);
}
```

```
/*-------------------------------------------------------------*/
/*                i_det4(a)                                    */
/*-------------------------------------------------------------*/

interval i_det4(a)              /* returns the determinant of   */
interval a[4][4];               /* the 4 x 4 matrix a           */
{
interval deti, detj, detk;
double i1, j1, k1;
int i, j, k, l;

i1 = 1.0;
deti = i_zero;
for(i = 0; i <= 3; i++)
{
   j1 = 1.0;
   detj = i_zero;
   for(j = 0; j <= 3; j++)
   {
      if(j == i)
         continue;
      k1 = 1.0;
      detk = i_zero;
      for(k = 0; k <= 3; k++)
      {
         if(k == j || k == i)
            continue;
            for(l = 0; l <= 3; l++)
            {
               if(l == k || l == j || l == i)
                  continue;
               detk = i_sum(detk, i_rmult(k1,
                     i_prod(a[l][0],a[k][1]) ) );
            }
         k1 = -k1;
      }
      detj = i_sum(detj, i_rmult(j1, i_prod(detk, a[j][2]) ) );
      j1 = -j1;
   }
   deti = i_sum(deti, i_rmult(i1, i_prod(detj, a[i][3]) ) );
   i1 = -i1;
}
return(deti);
}
```

```
/*
file: r_routines.h
*/

/*-------------------------------------------------------------------*/
/*                declarations                                       */
/*-------------------------------------------------------------------*/

double r_power_up();
double r_power_down();
double r_inv_up();
double r_inv_down();

double r_sum_up();
double r_sum_down();
double r_prod_up();
double r_prod_down();
double r_diff_up();
double r_diff_down();
```

```
/*
file: r_routines.c
*/

/*
routines in file:

1. r_power_up(r,k)      finds upper bound for r**k, r double,
                        k int, r >= 0, k >= 0 using the binary
                        expansion of k.

2. r_power_down(r,k)    finds lower bound for r**k, r double,
                        k int, r >= 0, k >= 0 using the binary
                        expansion of k.

3. r_inv_up(r)          finds upper bound for 1/r, r double, r != 0.
                        The routine uses the product routines
                        to check that the value returned can
                        be guarenteed to be an upper bound.

4. r_inv.down(r)        finds lower bound for 1/r, r double, r != 0.
                        The routine uses the product routines
                        to check that the value returned can
                        be guarenteed to be a lower bound.
*/

/*-------------------------------------------------------------------*/
/*                includes                                           */
/*-------------------------------------------------------------------*/

#include "r_macros.h"
#include "r_up.h"

/*-------------------------------------------------------------------*/
/*  1.            r_power_up(r,k)                                    */
/*-------------------------------------------------------------------*/

/* (1) r  < 0.0 || k < 0 ? if true print error message
   (2) s = 1.0
   (3) while ( k != 0) k even -> k = k/2, r = r**2
                       k odd  -> k = k-1, s = s*r
   (4) return s                                                     */
```

```
double r_power_up(r,k)
double r;
int k;
{
double s = 1.0;
if(r < 0.0 || k < 0)
   printf("error in r_power_up: k = %d, r = %e\n", k, r);
while(k > 0)
{
   if((k % 2) == 0)     /* k even                          */
   {
      k /= 2;
      r = r_prod_up(r,r);
   }
   else                 /* k odd                           */
   {
      k--;
      s = r_prod_up(r,s);
   }
}
return(s);
}

/*-------------------------------------------------------------------*/
/*  2.                  r_power_down(r,k)                            */
/*-------------------------------------------------------------------*/

/* (1) r  < 0.0  || k < 0? if true print error message
   (2) s = 1.0
   (3) while ( k != 0) k even -> k = k/2, r = r**2
                       k odd  -> k = k-1, s = s*r
   (4) return s                                                     */

double r_power_down(r,k)
double r;
int k;
{
double s = 1.0;
if(r < 0.0 || k < 0)
   printf("error in r_power_down: k = %d, r = %e\n", k, r);
while(k > 0)
{
   if((k % 2) == 0)     /* k even                          */
   {
      k /= 2;
      r = r_prod_down(r,r);
   }
   else                 /* k odd                           */
   {
      k--;
      s = r_prod_down(r,s);
   }
}
return(s);
}
```

```
/*-------------------------------------------------------------------*/
/*  3.                  r_inv_up(r,k)                                */
/*-------------------------------------------------------------------*/

/*
   (1) r == 0.0 ? if true print error message
   (2) s = r_up(1/r) first guess at upper bound
   (3) divide into two cases: Case (i)  r > 0
                              Case (ii) r < 0
   (4) increase s until upper bound is guarenteed
*/

double r_inv_up(r)
double r;
{
double s;               /* guess at upper bound of 1/r  */

if(r == 0.0)
{
   printf("error in r_inv_up: r = 0.0\n");
   exit(1);
}
s = r_up(1/r);          /* first guess at upper bound   */

if(r > 0)               /* r positive                   */
   while(r_prod_down(r,s) < 1.0)
      s = r_up(s);
                        /* increase s until we are sure */
                        /* we have an upper bound       */

else                    /* r negative                   */
   while(r_prod_up(r,s) > 1.0)
      s = r_up(s);
                        /* increase s until we are sure */
                        /* we have an upper bound       */
return(s);
}
```

```
/*-------------------------------------------------------------*/
/*  4.                  r_inv_down(r,k)                        */
/*-------------------------------------------------------------*/

/*
   (1) r == 0.0 ? if true print error message
   (2) s = r_down(1/r) first guess at lower bound
   (3) divide into two cases: Case (i)  r > 0
                              (ii) r < 0
   (4) decrease s until lower bound is guarenteed
*/

double r_inv_down(r)
double r;
{
double s;               /* guess at lower bound of 1/r  */

if(r == 0.0)
{
   printf("error in r_inv_down: r = 0.0\n");
   exit(1);
}

s = r_down(1/r);        /* first guess at lower bound   */

if(r > 0)               /* r positive                   */
   while(r_prod_up(r,s) > 1.0)
      s = r_down(s);
                        /* decrease s until we are sure */
                        /* we have a lower bound        */

else                    /* r negative                   */
   while(r_prod_down(r,s) < 1.0)
      s = r_down(s);
                        /* decrease s until we are sure */
                        /* we have a lower bound        */
return(s);
}
```

```
/*
file: rsp_routines.c
*/

/*-------------------------------------------------------------*/
/*                        includes                             */
/*-------------------------------------------------------------*/

#include "r_up.h"

/*-------------------------------------------------------------*/
/* 1.                  r_sum_up(r,s)                           */
/*-------------------------------------------------------------*/

/* (1) r == 0 ?  If true return s
   (2) s == 0 ?  if true return r
   (3) else return r_up(r+s)                                   */

double r_sum_up(r,s)
double r,s;
{
if(r == 0.0)
   return(s);
else if(s == 0.0)
   return(r);
else
   return(r_up(r+s));
}

/*-------------------------------------------------------------*/
/* 2.                  r_sum_down(r,s)                         */
/*-------------------------------------------------------------*/

/* (1) r == 0 ?  If true return s
   (2) s == 0 ?  if true return r
   (3) else return r_down(r+s)                                 */

double r_sum_down(r,s)
double r,s;
{
if(r == 0.0)
   return(s);
else if(s == 0.0)
   return(r);
else
   return(r_down(r+s));
}
```

```
/*-------------------------------------------------------------------*/
/* 3.                    r_prod_up(r,s)                              */
/*-------------------------------------------------------------------*/

/* (1) r == 0.0 ? If true return 0.0
   (2) s == 0.0 ? If true return 0.0
   (3) r == 1.0 ? If true return s
   (4) s == 1.0 ? If true return r
   (5) else return r_up(r*s)                                        */

double r_prod_up(r,s)
double r,s;
{
if(r == 0.0 || s == 0.0)
   return(0.0);
else if(r == 1.0)
   return(s);
else if(s == 1.0)
   return(r);
else
   return(r_up(r*s));
}

/*-------------------------------------------------------------------*/
/* 4.                    r_prod_down(r,s)                            */
/*-------------------------------------------------------------------*/

/* (1) r == 0.0 ? If true return 0.0
   (2) s == 0.0 ? If true return 0.0
   (3) r == 1.0 ? If true return s
   (4) s == 1.0 ? If true return r
   (5) else return r_down(r*s)                                      */

double r_prod_down(r,s)
double r,s;
{
if(r == 0.0 || s == 0.0)
   return(0.0);
else if(r == 1.0)
   return(s);
else if(s == 1.0)
   return(r);
else
   return(r_down(r*s));
}
```

```
/*-------------------------------------------------------------------*/
/* 5.                    r_diff_up(r,s)                              */
/*-------------------------------------------------------------------*/

/* (1) r == 0.0 ? If true return -s
   (2) s == 0.0 ? If true return  r
   (3) else return r_up(r - s)                                      */

double r_diff_up(r,s)
double r,s;
{
if(r == 0.0)
   return(-s);
else if(s == 0.0)
   return(r);
else
   return(r_up(r-s));
}

/*-------------------------------------------------------------------*/
/* 6.                    r_diff_down(r,s)                            */
/*-------------------------------------------------------------------*/

/* (1) r == 0.0 ? If true return -s
   (2) s == 0.0 ? If true return  r
   (3) else return r_down(r - s)                                    */

double r_diff_down(r,s)
double r,s;
{
if(r == 0.0)
   return(-s);
else if(s == 0.0)
   return(r);
else
   return(r_down(r-s));
}
```

```
/*
file: r_macros.h
*/

/*-------------------------------------------------------------------*/
/* 1.                    r_sum_up(r,s)                               */
/*-------------------------------------------------------------------*/

/* (1) r == 0 ?  If true return s
   (2) s == 0 ?  if true return r
   (3) else return r_up(r+s)                                        */

#define r_sum_up(r,s)   ((r) == 0.0 ? (s) : ((s) == 0.0 ? (r) : \
                        r_up((r) + (s)) ) )

/*-------------------------------------------------------------------*/
/* 2.                    r_sum_down(r,s)                             */
/*-------------------------------------------------------------------*/

/* (1) r == 0 ?  If true return s
   (2) s == 0 ?  if true return r
   (3) else return r_down(r+s)                                      */

#define r_sum_down(r,s) ((r) == 0.0 ? (s) : ((s) == 0.0 ? (r) : \
                        r_down((r) + (s)) ) )

/*-------------------------------------------------------------------*/
/* 3.                    r_prod_up(r,s)                              */
/*-------------------------------------------------------------------*/

/* (1) r == 0.0 ? If true return 0.0
   (2) s == 0.0 ? If true return 0.0
   (3) r == 1.0 ? If true return s
   (4) s == 1.0 ? If true return r
   (5) else return r_up(r*s)                                        */

#define r_prod_up(r,s)  ((r) == 0.0 ? 0.0 : ( (s) == 0.0 ? 0.0 : \
                        ((r) == 1.0 ? (s) : ( (s) == 1.0 ? (r) : \
                        r_up((r)*(s)) ) ) ) )

/*-------------------------------------------------------------------*/
/* 4.                    r_prod_down(r,s)                            */
/*-------------------------------------------------------------------*/

/* (1) r == 0.0 ? If true return 0.0
   (2) s == 0.0 ? If true return 0.0
   (3) r == 1.0 ? If true return s
   (4) s == 1.0 ? If true return r
   (5) else return r_down(r*s)                                      */

#define r_prod_down(r,s) ((r) == 0.0 ? 0.0 : ( (s) == 0.0 ? 0.0 : \
                        ((r) == 1.0 ? (s) : ( (s) == 1.0 ? (r) : \
                        r_down((r)*(s)) ) ) ) )

/*-------------------------------------------------------------------*/
/* 5.                    r_diff_up(r,s)                              */
/*-------------------------------------------------------------------*/

/* (1) r == 0.0 ? If true return -s
   (2) s == 0.0 ? If true return  r
   (3) else return r_up(r - s)                                      */

#define r_diff_up(r,s)          ( (r) == 0.0 ? -(s) : ( (s) == 0.0 \
                                ? (r) : r_up((r) - (s)) ) )

/*-------------------------------------------------------------------*/
/* 6.                    r_diff_down(r,s)                            */
/*-------------------------------------------------------------------*/

/* (1) r == 0.0 ? If true return -s
   (2) s == 0.0 ? If true return  r
   (3) else return r_down(r - s)                                    */

#define r_diff_down(r,s)        ( (r) == 0.0 ? -(s) : ( (s) == 0.0 \
                                ? (r) :  r_down((r) - (s)) ) )

/*-------------------------------------------------------------------*/
/*              procedure declarations                               */
/*-------------------------------------------------------------------*/

double r_power_up();
double r_power_down();
double r_inv_up();
double r_inv_down();
```

```
/*
file: r_up.h
*/

/*-------------------------------------------------------------*/
/*              declarations                                   */
/*-------------------------------------------------------------*/

double r_up();
double r_down();
```

```
/*
file: r_up.c
*/

/*-------------------------------------------------------------*/
/*              r_up(r)                                        */
/*-------------------------------------------------------------*/

double r_up(r)
double r;
{
static double rtemp = 0.0;      /* temporary variable that contains
                                   the increment to be added to r
                                 */
static int *p = (int *) &rtemp; /* integer pointer used to place
                                   the exponent bits of rtemp in
                                   the correct position
                                 */
*p = (*((int *) &r) & 0x00007f80) - 0x00001b80;
                                /* access the exponent of r by
                                   bitwise & with mask, subtract
                                   integer variable corresponding
                                   to exponent 55, and place in
                                   position in rtemp
                                 */
if(*p <= 0) *p = 0x00000080;    /* if resulting exponent  <= 0
                                   put the smallest normalisable
                                   number in rtemp
                                 */
return(r + rtemp);              /* add rtemp to r               */
}

/*-------------------------------------------------------------*/
/*              r_down(r)                                      */
/*-------------------------------------------------------------*/

double r_down(r)
double r;
{
static double rtemp = 0.0;      /* temporary variable that contains
                                   the increment to be subtracted
                                   from r
                                 */
static int *p = (int *) &rtemp; /* integer pointer used to place
                                   the exponent bits of rtemp in
                                   the correct position
                                 */
*p = (*((int *) &r) & 0x00007f80) - 0x00001b80;
                                /* access the exponent of r by
                                   bitwise & with mask, subtract
                                   integer variable corresponding
                                   to exponent 55, and place in
                                   position in rtemp
                                 */
```

```
if(*p <= 0) *p = 0x00000080;    /* if resulting exponent <= 0
                                   put the smallest normalisable
                                   number in rtemp
                                */
return(r - rtemp);              /* subtract rtemp from r          */
}
```

```
/*
file: print.c
*/

/*-------------------------------------------------------------*/
/*              print.c                                        */
/*              This file contains a variable                  */
/*              argument routine that generalises              */
/*              printf.                                        */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*              includes                                       */
/*-------------------------------------------------------------*/

#include <stdio.h>
#include <varargs.h>
#include <strings.h>
#include <math.h>

/*-------------------------------------------------------------*/
/*              print(a1,a2,...)                               */
/*-------------------------------------------------------------*/

void print(va_alist)
va_dcl
{
va_list ap;
void prquot();
char *format,*cp,a[100],b[2],c[100];

va_start(ap);
strcpy(a,va_arg(ap,char *));
format = a;
strcpy(b," ");
strcpy(c,"%");
cp = format;
while((cp = index(format,'%')) != NULL)
{
   *cp = '\0';
   printf(format);
   format = ++cp;

again:
   switch((*b = *cp))
   {

      /* special option */
      case 'q':
      {
         prquot(va_arg(ap,double));
         break;
      }
```

```
/* normal options */

case 's':
{
   *cp = '\0';
   strcpy(c+1,format);
   printf(strcat(c,b),va_arg(ap,char *));
   break;
}

case 'c':
case 'x':
case 'd':
case 'o':
{
   *cp = '\0';
   strcpy(c+1,format);
   printf(strcat(c,b),va_arg(ap,int));
   break;
}

case 'e':
case 'g':
case 'f':
{
   *cp = '\0';
   strcpy(c+1,format);
   printf(strcat(c,b),va_arg(ap,double));
   break;
      }

      case '\0':
      {
         printf("%s",format);
         return;
      }
      default:
      {
         cp++;
         goto again;
      }
   }
   format = ++cp;
}
printf(format);
return;
}
```

```
/*-------------------------------------------------------------*/
/*                    prquot(a)                                */
/*-------------------------------------------------------------*/

void prquot(a)
double a;
{
double b,big,f,g,h,modf();
int i,exp,expg,count;

big = ldexp(1.0,55);
if(a == 0.0)                          /* special case a = 0  */
{
   printf("%d",0);
   return;
}
if(a < 0)                             /* if a < 0, print out */
{                                     /* minus sign and put  */
   printf("-");                       /* a = -a              */
   a = -a;
}
if(a > big)                           /* |a| too big         */
{
   printf("\nprquot:ERROR: |a| too big, |a| = %e",a);
   return;
}
a = modf(a,&b);                       /* b = integer  part a */
                                      /* a = fraction part a */

if(b > 0)                             /* print out int part  */
   printf("%.0f ",b);                 /* if not 0            */

if(a == 0.0)                          /* if fraction = 0     */
   return;                            /* return              */

f = frexp(a,&exp);                    /* a = f*2^exp, with   */
                                       /* 0 <= f <= 0.5       */

count = 0;                            /* count the number of */
g = f;                                /* trailing zeros of f */
for(i = 0; i <= 56; i++)
{
   if(g == 0.0)                       /* all the rest of f 0 */
   {
      count = i;
      break;
   }
   h = frexp(g, &expg);               /* double g !!!        */
   g = ldexp(h, expg + 1);
   g = modf(g, &b);                   /* g = frac part of g  */
}
                                      /* print out f in rat  */
                                      /* form                */
printf("%.0f/2^%d",ldexp(f,count),count - exp);

return;
}
```

```
Program to Prove the Existence of a Fixed Point of the
-------------------------------------------------------
Renormalisation Transformation for Cubic Critical Maps
-------------------------------------------------------
of the Circle wth Golden Mean Rotation Number.
-------------------------------------------------------
n, deg of polynomial part = 80
nh = 81, nk = 163
centres and radii of domains:
exact values for domains:
a1 = -63457/2^18
r1 = 9898557/2^25
a2 = 4344147/2^23
r2 = 9898557/2^24
approximate values for domains:
a1 = -2.420692e-01
r1 = 2.950000e-01
a2 = 5.178627e-01
r2 = 5.900000e-01
-----------------------------------
basis change matrix invertible
-----------------------------------
norm of N(h0, k0) - (h0, k0) bounded by 50078928353969305/2^96,
which is approx. 6.320849e-13
-----------------------------------
radius of ball around (h0, k0) = 6259866042461631/2^93,
which is approx. 6.320849e-12
radius about h0 (standard basis) = 13633774479461695/2^87,
which is approx. 8.810620e-11
radius about k0 (standard basis) = 7800634234722533/2^84,
which is approx. 4.032833e-10
-----------------------------------
i_dt:analyticity improvement factors:
norm of sbx1 = 9.196636e-01
norm of sbx2 = 9.348640e-01
norm of shb3 = 9.772217e-01
dt is a compact operator
-----------------------------------
i_dtm4: contracted derivative matrix(decimal approx):
i_dtm4[0][0] = [-2.8336110525911624e+00, -2.8336100204852128e+00]
i_dtm4[0][1] = [-2.9480361856146717e-04, -2.9371125048468131e-04]
i_dtm4[0][2] = [ 6.1868521539655040e-04,  6.2002035872793928e-04]
i_dtm4[0][3] = [-1.3648414216490633e-03,  1.3648414216490633e-03]
i_dtm4[1][0] = [-1.6169395587312669e-04, -1.6023434839602259e-04]
i_dtm4[1][1] = [ 2.1395816174205921e+00,  2.1395831702566162e+00]
i_dtm4[1][2] = [ 9.0575549461714500e-04,  9.0766694292685952e-04]
i_dtm4[1][3] = [-1.6702256105482754e-03,  1.6702256105482754e-03]
i_dtm4[2][0] = [ 3.5092929817840734e-05,  3.6666508595856077e-05]
i_dtm4[2][1] = [-3.1313627300563176e-05, -2.9644868264691l9e-05]
i_dtm4[2][2] = [-1.0000021527226788e+00, -1.0000001075978891e+00]
i_dtm4[2][3] = [-1.0306335499548177e-02,  1.0306335499548177e-02]
i_dtm4[3][0] = [-1.8652581773301330e-02,  1.8652581773301330e-02]
i_dtm4[3][1] = [-2.4860675554301707e-02,  2.4860675554301707e-02]
i_dtm4[3][2] = [-2.9901763064798242e-02,  2.9901763064798242e-02]
i_dtm4[3][3] = [-6.9770622060129091e-01,  6.9770622060129091e-01]
-----------------------------------
N is a contraction on ball around (h0,k0)
T has a unique fixed point in ball
-----------------------------------
circle 1:
centre = -2 14411518807585587/2^54,
which is approx. -2.800000e+00
radius = 57646075230342349/2^59,
which is approx. 1.000000e-01
circle 2:
centre = -1 ,
which is approx. -1.000000e+00
radius = 57646075230342349/2^59,
which is approx. 1.000000e-01
circle 3:
centre = 0,
which is approx. 0.000000e+00
radius = 7/2^3,
which is approx. 8.750000e-01
circle 4:
centre = 2 1261007895663739/2^53,
which is approx. 2.140000e+00
radius = 57646075230342349/2^59,
which is approx. 1.000000e-01
-----------------------------------
determinant changes sign as expected
-----------------------------------
beta is contained in [-27960195960041105/2^55, -873756122843171/2^50],
which is approx. [-7.760513331967229Se-01, -7.760513323901561le-01]
-----------------------------------
i_comb = [31481284377326817/2^57, 62962570218246731/2^58]
which is approx. [ 2.1844529225300292e-01,  2.1844529733086153e-01]
i_dg3 = [-2 1257230788240771/2^53, -2 1257229876927719/2^53]
which is approx. [-2.1395806568372538e+00, -2.1395805556611805e+00]
g(b^6)^3 = b^6
Dg(b^6) < 0
-----------------------------------
h has positive derivative at 0
k has positive derivative at 0
-----------------------------------
eigenvalue no 0 =approx -2.8336106558912252e+00
norm of Ni(lam',u') - (lam',u') = 3.528799e-05
norm of derivative = 2.578907e-01
Ni is a contraction on ball
eigenvalue 0 contained in [-2 15023351476140371/2^54, -2 15010637637810001/2^54]
which is approx. [-2.8339635357923686e+000, -2.8332577759900817e+00]
-----------------------------------
eigenvalue no 1 =approx  2.1395806062492500e+00
norm of Ni(lam',u') - (lam',u') = 2.180643e-05
norm of derivative = 3.357929e-01
Ni is a contraction on ball
eigenvalue 1 contained in [2 2510532367133243/2^54, 2 2518388963204919/2^54]
which is approx. [ 2.1393625419029011e+00,  2.1397986705955989e+00]
eigenvector number 1 violates dF*(0)(dh,dk) = 0
```

```
----------------------------------
eigenvalue no 2 =approx -1.0000000000000049e+00
norm of Ni(lam',u') - (lam',u') = 1.076838e-04
norm of derivative = 7.639265e-01
Ni is a contraction on ball
eigenvalue 2 contained in [-1 9699296402631/2^53, -71979999666707597/2^56]
which is approx. [-1.0010768382188865e+00, -9.9892316178112335e-01]
eigenvector number 2 violates dF*(3)(dh,dk) = 0
```

A8. Error bounding for VAX-11/750 Computers, Interval Arithmetic, and Function Ball/Vector Arithmetic

This appendix contains details of the error bounding routines implemented on the VAX-11/750 computer. This is followed by a detailed description of the interval arithmetic operations and their computer implementation. Finally, we describe in detail the Function Ball/vector operations.

A8.1 Error Bounding

A8.1.1 Floating Point Representation and the functions r up and r down.

Double variables in C are implemented on the VAX-11/750 computer as D-floating data as defined in the VAX Architecture Guide. The D-floating datum consists of 8 contiguous bytes of storage, 64 bits in all. The bits are labelled from 0 to 63. 55 bits form the fraction, and there is an 8 bit exponent and a sign bit. Table A8.1 is an extract from the VAX Architecture Guide that describes the D-Floating datum and the location of the bits corresponding to the fraction, the exponent and the sign bit. The VAX uses a "hidden bit" in its floating point representation. The fraction bits represent a number between ½ and 1. All the fraction bits equal to 0 corresponds to a fraction ½ rather than 0. The 8 bit exponent ranges from 0 to 255. Floating point 0 is represented by all the exponent bits equal to 0 (whatever the fraction, although a zero produced by a floating point operation has all bits set to 0). The other 255 values represent binary exponents of -127 to 127 inclusive, 1 corresponding to -127 and 255 corresponding to 127.

The set R of numbers that can be represented as D-floating data on

• F_floating (single-precision floating)

An F_floating datum, sometimes called just "floating" or "single-precision floating," is four contiguous bytes starting on an arbitrary byte boundary or in register n. The bits are labelled from the right 0 through 31.

The form of an F_floating datum is sign magnitude with bit 15 the sign bit, bits 14:7 an excess 128 binary exponent, and bits 6:0 and 31:16 a normalized 24-bit fraction with the redundant most significant fraction bit not represented. Within the fraction, bits increase in significance from 16 through 31 and 0 through 6. The 8-bit exponent field encodes the values 0 through 255. An exponent value of 0 together with a sign bit of 0, indicates that the F_floating datum has a value of 0. Exponent values of 1 through 255 indicates true binary exponents of −127 through +127. An exponent value of 0, together with a sign bit of 1, is taken as reserved. (Floating point instructions processing a reserved operand take a reserved operand fault.) The magnitude of an F_floating datum is in the approximate range $.29*10^{-38}$ through $1.7*10^{38}$. The precision of an F_floating datum is approximately one part in 2^{23} (approximately 7 decimal digits.)

• D_floating (double-precision floating)

The D_floating datum sometimes referred to as "double floating" or "double-precision floating," is eight contiguous bytes starting on an arbitrary byte boundary or in two consecutive registers, R[n+1]'R[n]. The bits are labelled from the right 0 through 63.

The form of a D_floating datum is identical to the F_floating datum except for an additional 32 low significance fraction bits. Within the fraction, bits increase in significance from 48 through 63, 32 through 47, 16 through 31, and 0 through 6, as suggested by the widening arrow in Figure 4-2. The exponent conventions, and approximate range of values is the same for both D_floating and F_floating. The precision of a D_floating datum is approximately one part in 2^{55} (approximately 16 decimal digits).

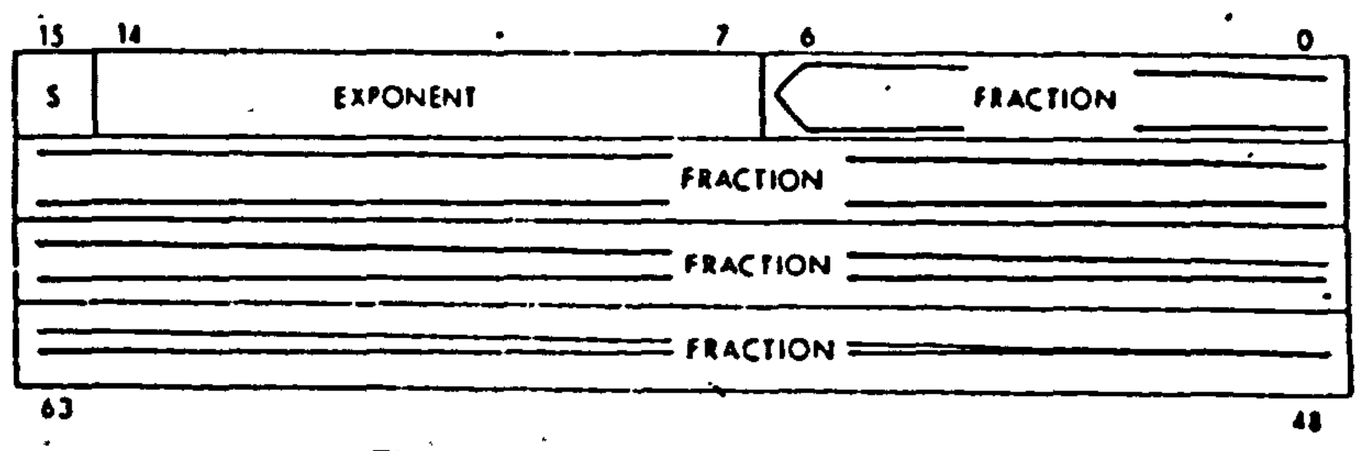


Figure 4-2 D_floating Format

Table A8.1 Extract from the VAX Architecture Guide detailing the D-datum/double variable storage (Pages 34-35)

VAX-11 computers is precisely:

$$\{0\} \cup \{ \pm f.2^k \},\ -127 < k < 127,\ f = \ell/2^{56},\ 2^{55} < \ell < 2^{56}$$

We note that C int variables are implemented as longword data comprising four contiguous bytes of storage.

The functions r_up and r_down of Chapter 4 are implemented in C as the double functions r_up and r_down. The two functions r_up and r_down are similar. We shall describe r_up. r_down(r) may be defined as -r_up(-r). The source code for r_up is the following:

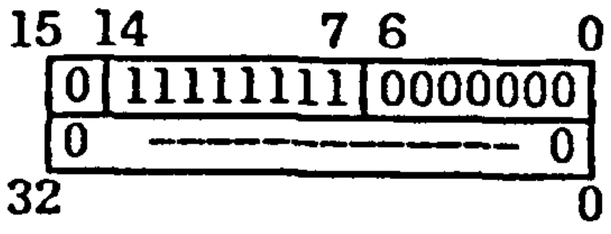
```
/*-------------------------------------------------------------*/
/*                      r_up(r)                                */
/*-------------------------------------------------------------*/

double r_up(r)
double r;
{
 static double rtemp = 0.0;        /* temporary variable that contains
                                      the increment to be added to r
                                   */
 static int *p = (int *) &rtemp; /* integer pointer used to place
                                      the exponent bits of rtemp in
                                      the correct position
                                   */
 *p = (*((int *) &r) & 0x00007f80) - 0x00001b80;
                                  /* access the exponent of r by
                                      bitwise & with mask, subtract
                                      integer variable corresponding
                                      to exponent 55, and place in
                                      position in rtemp
                                   */
 if(*p <= 0) *p = 0x00000080;     /* if resulting exponent  <= 0
                                      put the smallest normalisable
                                      number in rtemp
                                   */
 return(r + rtemp);               /* add rtemp to r                */
 }
```

We denote by r^+ the number min { s ∈ R : s > r }. Then r^+ is the smallest number greater than r that can be represented as a D-floating datum on the VAX-11/750. Likewise we set r^- = max { s ∈ R : s < r}. Now, r_up(r) returns a number in R that is slightly larger than r. Usually r_up(r) = r^+. There are two exceptions to this rule: (1) when | r | < 2^{-73} and (2) when r = -2^n, for some integer n. We shall discuss these two cases below. In all cases, r_up(r) is greater than r and difffers by at least one Least Significant Bit (L.S.B.) of r.

The precise allocation of bits for D-floating data is displayed in the figure in Table (A8.1). A precise mathematical definition of r_up is the following:

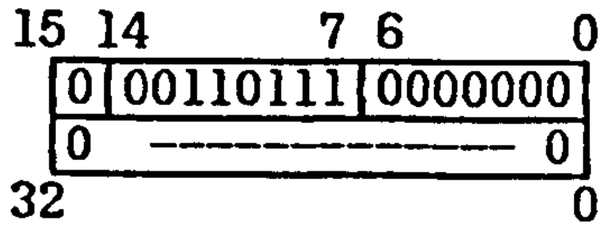
$$r\_up:\ r = \pm f.2^k \longrightarrow r + \tfrac{1}{2}.2^{\max(-127,\ k\ -\ 55)}$$

r_up is implemented in C by adding to r a double variable that corresponds to a number with a one in the L.S.B. position of r. Because of the hidden bit, this merely requires extracting the exponent of r, subtracting the precision of the fraction, placing the result in a double variable with fraction bits all set to 0 and adding the result to r. This addition is performed exactly by the computer. We refer to the code for r_up. rtemp is the variable that contains the number to be added to r. It is set to 0 on entering the function. p is an integer pointer that points to the first 32 bits of rtemp. The exponent is obtained by taking the first 32 bits of r (i.e. *((int *) &r))) and masking this with the integer variable 0x00007f80 which has bit pattern

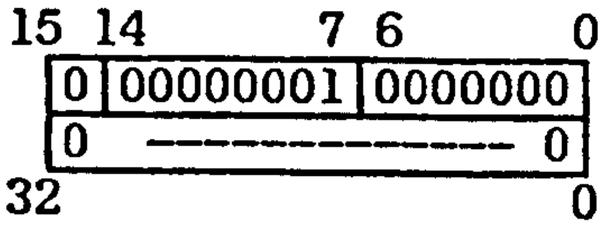


Then 55 is subtracted from the exponent by subtracting the integer

constant 0x00001b80 which has bit pattern



corresponding to 55 in the exponent bit field. The result is then placed in p which points to the first 32 bits of rtemp. Now if *p < 0, then the exponent of rtemp is less than or equal to -128 and so we replace rtemp by the smallest positive number in R. This has exponent -127 and an all-zero fraction bit pattern. The first 32 bits have hexadecimal representation 0x00000080 and bit pattern



The last 32 bits are all 0. After rtemp has been calculated, it is added to r and the result is returned as the value of r_up.

We consider briefly the two exceptional cases when $r\_up(r) \neq r^+$.

Case (1) $r = \pm f.2^k$, $\frac{1}{2} < f < 1$, $-127 < k < -73$. In the cases r_up(r) is greater than r^+. $r^+ = r + \frac{1}{2}.2^{k-55}$, while $r\_up(r) = r + \frac{1}{2}.2^{-127}$. This discrepancy occurs because we have used the floating point operations to obtain r_up(r). $\frac{1}{2}.2^{-127}$ is the smallest positive number that can be added to a number in R.

Case (2) $r = -\frac{1}{2}.2^k$, $-72 < k < 127$. In this case $r\_up(r) = r + \frac{1}{2}.2^{k-55}$ $= -(2^{55} - 1)/2^{55}.2^{k-1}$ while $r^+ = -(2^{56} - 1)/2^{56}.2^{k-1}$. In fact this slight deviation is necessary for r_up(r) to differ from r by at least 1 L.S.B. of r (r^+ differs from r by ½ L.S.B. of r).

A8.1.2 Vax Floating Point Arithmetic

The arithmetic routines for double variables in C are translated into the VAX floating point operations on D-floating data by the C compiler. Table A8.2 contains an extract from the VAX Architecture

APPENDIX H
ACCURACY

It will now be shown that an overflow bit and two guard bits are adequate to guarantee accuracy of rounded ADD, SUB, MUL, or DIV, provided, of course, that the algorithms are properly chosen. Note, first, that ADD and SUB may result in propagation of a carry, and hence the overflow bit is necessary. Second, if in ADD or SUB there is a one-bit loss of significance in conjunction with an alignment shift of two or more bits, the first guard bit is needed for the LSB of the normalized result, and the second is then the rounding bit. So the three bits are necessary. A number of constraints must be observed in selection of the algorithms for the basic operations in order for these three bits to be sufficient to guarantee an error bound of (½) LSB:

1. ADD or SUB:
   - If the alignment shift does not exceed 2 there are no constraints, because no bits can be lost.
   - If the alignment shift exceeds 2 (or however many guard bits are used, say g ≥ 2), no negations may be made after the aligment shift takes place.
   - If the above constraint is observed, the error bound for a rounded result is (½) LSB. If, however, a negation follows the alignment shift, the error bound will be

     (½) * (1 + 2**(−g+2)) LSB

     because a "borrow" will be lost on an implicit subtraction, if nonzero bits were lost in the alignment shift. Note that the error bound is 1 LSB if the constraint is ignored and there are only two guard bits (g = 2).
   - The constraint on no negations after the alignment shift may be replaced by keeping track of nonzero bits lost during the alignment shift, and then negating by one's complement if any "ones" were lost, and by two's complement if none were lost. If this is done, the error bound will be (½) LSB.
2. MUL:
   - The product of two normalized binary fractions can be as small as ¼ and must be less than one. The overflow bit is not needed for MUL, but the first guard bit will be necessary for normalization if the product if less than ½, and, in this case, the second guard bit is the rounding bit.

   - The first constraint on MUL is that the product be generated from the least to the most significant bit. Low order bits, in positions to the right of the second guard bit, may be discarded, but ONLY AFTER they have made their contribution to carries which could propagate into the guard bits or beyond.
   - For the same reasons as for ADD or SUB, if low order bits of the product have been discarded, no negations can be made after generating the product.
3. DIV:
   - For standard algorithms it is necessary that the remainder be generated exactly at each step; the overflow and two guard bits are adequate for this purpose. The register receiving the quotient must have a guard bit for the rounding bit, and the quotient must be developed to include the rounding bit.
   - The Newton-Raphson quadratic convergence algorithms require a number of guard bits equal to twice the number of bits desired in the result if the correctness of the rounding bit is to be guaranteed.

VAX observes all constraints and generates floating point results with an error bound of (½) LSB for all floating instructions except EMOD and POLY (see EMOD and POLY descriptions.)

Table A8.2 Extract from the VAX Architecture Guide describing the accuracy of the floating point instructions. (Pages 469-470)

Guide. The guide states that all VAX floating point instructions (except two which we do not use) produce results within an error margin of ½L.S.B. Therefore, since r_up(r) differs from r by at least one L.S.B. of r, we know that for $\odot \in \{ +, -, \times, \div \}$, r, s ∈ R, we have $r \odot s < r\_up(r \odot_C s)$, where $r \odot_C s$ is the result produced by the VAX instruction corresponding to ⊙. In fact in order to make the routines as portable as possible we have included an extra check in the case of division. Because division algorithms vary so greatly, it is best to check explicitly an upper bound for the result of a division instruction. This is done using the multplication instruction and is described below. The comments of the last paragraph apply mutadis mutandis to the function r_down.

We provide the following error bounding routines. Let r, s ∈ R.

r_sum_up(r,s)	returns upper bound for r + s
r_sum_down(r,s)	returns lower bound for r + s
r_prod_up(r,s)	returns upper bound for r × s
r_prod_down(r,s)	returns lower bound for r × s
r_diff_up(r,s)	returns upper bound for r - s
r_diff_down(r,s)	returns lower bound for r - s
r_power_up(r,k)	returns upper bound for r^k, r, k > 0
r_power_down(r,k)	returns lower bound for r^k, r, k > 0
r_inv_up(r,s)	returns upper bound for 1 ÷ r, r ≠ 0
r_inv_down(r,s)	returns lower bound for 1 ÷ r, r ≠ 0

The functions r_sum_up, r_sum_down, r_prod_up, r_prod_down, r_diff_up, r_diff_down may be readily implemented as macros, that is, routines that are implemented at compile time. These routines use the functions r_up and r_down described above to obtain upper and lower bounds. With the exception of special cases, the functions

r_sum_up, r_sum_down, r_prod_up, r_prod_down, r_diff_up, r_diff_down are defined in terms of the VAX floating point operations $+_C$, $\times_C$, $-_C$ as

r_sum_up(r,s) = r_up(r $+_C$ s) r,s ≠ 0

r_sum_down(r,s) = r_down(r $+_C$ s) r,s ≠ 0

r_prod_up(r,s) = r_up(r $\times_C$ s) r,s ≠ 0, 1

r_prod_down(r,s) = r_down(r $\times_C$ s) r,s ≠ 0, 1

r_diff_up(r,s) = r_up(r $-_C$ s) r,s ≠ 0

r_diff_down(r,s) = r_down(r $-_C$ s) r,s ≠ 0

For the exceptional cases the exact result is returned. The routine r_inv_up(r), r ≠ 0, takes r_up(1 $+_C$ r) as an initial guess for an upper bound s. It checks that s can be guarenteed to be an upper bound by use of the multiplication routines r_prod_up and r_prod_down. We suppose r > 0 (the case when r < 0 is analogous). Then, if s satisfies r_prod_down(r,s) > 1, we know that s is an upper bound for 1 + r. If s fails this test we put s = r_up(s) and test again. We continue this process until s is guarenteed to be an upper bound for 1 + r. This may seem elaborate but on some computers (although not the VAX-11) this checking is necessary to ensure that the upper bound is rigorous. The routine r_inv_down is similar.

The routines r_power_up(r,k) and r_power_down(r,k), r,k > 0, use the routines r_prod_up and r_prod_down respectively to calculate upper and lower bounds for r^k. The algorithm for calculating r^k is standard based on the binary expansion of k (see e.g. Knuth (1969)).

A8.1.3 Floating Point Overflow

In this section we describe briefly how the VAX running UNIX deals with floating point overflows.

The manner in which UNIX handles floating point exceptions is

discussed in the signal(2) section of the Berkeley UNIX manual. A signal from a floating point exception causes "termination of the receiving process" i.e. the program is halted.

A floating point exception is caused by a floating point overflow as described in section 6.4.1.8 of the VAX Architecture Manual. This occurs when an instruction results in "an exponent greater than the largest representable exponent for the data type after normalization and rounding." Table A8.3 contains a brief extract from the Manual detailing the floating point exception.

We see that floating point overflow will result in program termination should it occur during the running of the program.

A8.2 Interval Arithmetic

In Chapter 4 the basic principle of interval arithmetic is described. We list the exact interval arithmetic routines together with their computer implementation. The standard reference for interval arithmetic is Moore (1966). Let $I_1 = [a_1, b_1]$ and $I_2 = [a_2, b_2]$ be finite closed intervals in $\mathbb{R}$. We define the following exact interval arithmetic operations:

(1) addition, sum $\quad I_1 +_i I_2 = [a_1 + a_2, b_1 + b_2]$

(2) negation, unary minus $\quad -_i I_1 = [-b_1, -a_1]$

(3) subtraction, difference $\quad I_1 -_i I_2 = [a_1 - b_2, b_1 - a_2]$

(4) real multiplication $\quad r \times_i I_1 = [r.a_1, r.b_1]$ if $r \geq 0$; $[r.b_1, r.a_1]$ if $r < 0$; $r \in \mathbb{R}$

(5) multiplication, product $\quad I_1 \times_i I_2 = [a_3, b_3]$ where

$a_3 = \min\{a_1.a_2, a_1.b_2, b_1.a_2, b_1.b_2\}$

$b_3 = \max\{a_1.a_2, a_1.b_2, b_1.a_2, b_1.b_2\}$

pushed on the stack is 6 (SRMSK_DEC_OVF_T).

6.4.1.7 Subscript Range Trap - A subscript range trap is an exception that indicates that the last instruction was an INDEX instruction with a subscript operand that failed the range check. The value of the subscript operand is lower than the low operand or greater than the high operand. The result is stored in indexout, and the condition codes are set as if the subscript were within range. The type code pushed on the stack is 7 (SRMSK_SUB_RNG_T).

6.4.1.8 Floating Overflow Fault - A floating overflow fault is an exception that indicates that the last instruction executed resulted in an exponent greater than the largest representable exponent for the data type after normalization and rounding. The destination was unaffected and the saved condition codes are UNPREDICTABLE. The saved PC points to the instruction causing the fault. In the case of a POLY instruction, the instruction is suspended with FPD set (see Chapter 4 for details). The type code pushed on the stack is 8 (SRMSK_FLT_OVF_F).

6.4.1.9 Divide By Zero Floating Fault - A floating divide by zero fault is an exception that indicates that the last instruction executed had a floating zero divisor. The quotient operand was unaffected and the saved condition codes are UNPREDICTABLE. The saved PC points to the instruction causing the fault. The type code pushed on the stack is 9 (SRMSK_FLT_DIV_F).

6.4.1.10 Floating Underflow Fault - A floating underflow fault is an exception that indicates that the last instruction executed resulted in an exponent less than the smallest representable exponent for the data type after normalization and rounding and that floating underflow was enabled (FU set). The destination operand is unaffected. The saved condition codes are UNPREDICTABLE. The saved PC points to the instruction causing the fault. In the case of a POLY instruction, the instruction is suspended with FPD set (see Chapter 4 for details). The type code pushed on the stack is 10 (SRMSK_FLT_UND_F).

Table A8.3 Extract from the Vax Architecture Manual detailing the floating overflow fault (6.4.1.8) (page 6-16).

(6) inversion $1 \div_i I_1 = [1/b_1, 1/a_1],\ 0 \notin I_1$

(7) division, quotient $I_1 \div_i I_2 = I_1 \times_i (1 \div_i I_2) ,\ 0 \notin I_2.$

(8) absolute value (a) $abs_r(I_1) = \max\{ |a_1|, |b_1| \}$

(9) absolute value (b) $abs_i(I_1) = [a_3, b_3]$ where

(i) if $0 \in I_1$, $a_3 = 0$, $b_3 = abs_r(I_1)$

(ii) if $0 \notin I_1$, $a_3 = \min\{|a_1|, |b_1|\}$

$b_3 = \max\{|a_1|, |b_1|\}$

These definitions of exact interval arithmetic operations all satisfy equations corresponding to equation (4.3) with equality.

An interval is implemented in C as the following structure:

```
typedef struct {
        double lo;  /* left  hand end-point of interval   */
        double up;  /* right hand end-point of interval   */
} interval;
```

There are three global interval constants:

i_zero - the zero interval [0, 0]

i_one - the interval [1, 1]

i_unit - the unit interval [-1, 1]

The following are the interval routines that are implemented:

i_intr, i_intrl, i_inti, i_intil, r_av, i_neg, i_sum, i_diff, i_prod, i_inv, i_quot, i_rmult, i_power, i_abs, r_abs, int_zero, i_matmult, i_matvec, r_matcol, i_matmult4, i_matvec4, r_matcol4, i_det4. They are contained within the file i_routines.c. The interval typedef statement and the declarations for these routines are contained in the file i_routines.h. The following is a brief description of each of these routines. The statements made are all simple exercises with inequalities.

1. interval i_intr(r,s)

   double r, s;

This routine returns the interval [r, s].

2. interval i_intrl(r)

   double r;

This routine returns the interval [r, r].

3. interval i_inti(i,j)

   int i,j;

This routine returns the interval [i, j], with i,j converted to double.

4. interval i_intil(i)

   int i;

This routine returns the interval [i, i], with i converted to double.

5. double r_av(a)

   interval a;

This routine returns (an approximation to) the middle point of the interval a.

6. interval i_neg(a)

   interval a;

This routine returns the negative of the interval a. This is simply [-a.up, -a.lo].

7. interval i_sum(a,b)

   interval a, b;

This routine returns an interval c containing a $+_i$ b. The precise

definition of c is:

c.lo = r_sum_down(a.lo, b.lo) $\leq$ a.lo + b.lo

c.up = r_sum_up (a.up, b.up) $\geq$ a.up + b.up

so that a $+_i$ b $\subseteq$ c.

8. interval i_diff(a,b)

interval a,b;

This routine returns an interval c containing a $-_i$ b. The precise definition of c is:

c.lo = r_diff_down(a.lo, b.up) $\leq$ a.lo - b.up

c.up = r_diff_up (a.up, b.lo) $\geq$ a.up - b.lo

so that a $-_i$ b is contained in c.

9. interval i_rmult(r,a)

double r; interval a;

This routine returns an interval c containing r $\times_i$ a = {r.x : x $\in$ a}.

The precise definition for c is:

If r = 0, then c == [0,0].

If r > 0, then c.lo = r_prod_down(r, a.lo) $\leq$ r.x for all x $\in$ a,

and c.up = r_prod_up (r, a.up) $\geq$ r.x for all x $\in$ a.

If r < 0, then c.lo = r_prod_down(r, a.up) $\leq$ r.x for all x $\in$ a,

and c.up = r_prod_up (r, a.lo) $\geq$ r.x for all x $\in$ a.

Thus c contains the interval r $\times_i$ a.

10. interval i_prod(a,b)

interval a,b;

This routine returns an interval c containing a $\times_i$ b. The definition (5) of the exact interval product may be readily implemented on the computer as

c.lo = r_down(min(a.lo*b.lo,a.lo*b.up,a.up*b.lo,a.up*b.up))

c.up = r_up (max(a.lo*b.lo,a.lo*b.up,a.up*b.lo,a.up*b.up))

However, in order to reduce the number of multiplications that will be required in most cases to two, we distinguish nine separate cases:

(1) $0 \leqslant a.lo \leqslant a.up,\ 0 \leqslant b.lo \leqslant b.up$

Then

$a.lo \times b.lo \leqslant a.lo \times b.up,\ a.up \times b.lo$

$a.up \times b.up \geqslant a.lo \times b.up,\ a.up \times b.lo$

so we put

c.lo = r_prod_down(a.lo, b.lo)

c.up = r_prod_up (a.up, b.up).

(2) $0 \leqslant a.lo \leqslant a.up,\ \ b.lo \leqslant b.up \leqslant 0$

Then

$a.up \times b.lo \leqslant a.lo \times b.lo,\ a.up \times b.up$

$a.lo \times b.up \geqslant a.lo \times b.lo,\ a.up \times b.up$

so we put

c.lo = r_prod_down(a.up, b.lo)

c.up = r_prod_up (a.lo, b.up).

(3) $0 \leqslant a.lo \leqslant a.up,\ \ b.lo \leqslant 0 \leqslant b.up$

Then

$a.up \times b.lo \leqslant a.lo \times b.lo \leqslant 0 \leqslant a.lo \times b.up \leqslant a.up \times b.up$

so we put

c.lo = r_prod_down(a.up, b.lo)

c.up = r_prod_up (a.up, b.up).

(4) $a.lo \leqslant a.up \leqslant 0,\ 0 \leqslant b.lo \leqslant b.up$

Then

$a.lo \times b.up \leqslant a.lo \times b.lo,\ a.up \times b.up$

$a.up \times b.lo \geqslant a.lo \times b.lo,\ a.up \times b.up$

so we put

c.lo = r_prod_down(a.lo, b.up)

c.up = r_prod_up (a.up, b.lo).

(5) a.lo ≤ a.up ≤ 0, b.lo ≤ b.up ≤ 0

Then

a.up × b.up ≤ a.lo × b.up, a.up × b.lo

a.lo × b.lo ≥ a.lo × b.up, a.up × b.lo

so we put

c.lo = r_prod_down(a.lo, b.up)

c.up = r_prod_up (a.lo, b.lo).

(6) a.lo ≤ a.up ≤ 0, b.lo ≤ 0 ≤ b.up

Then

a.lo × b.up ≤ a.up × b.up ≤ 0 ≤ a.up × b.lo ≤ a.lo × b.lo

so we put

c.lo = r_prod_down(a.lo, b.up)

c.up = r_prod_up (a.lo, b.lo).

(7) a.lo ≤ 0 ≤ a.up, 0 ≤ b.lo ≤ b.up

Then

a.lo × b.up ≤ a.lo × b.lo ≤ 0 ≤ a.up × b.lo ≤ a.up × b.up

so we put

c.lo = r_prod_down(a.lo, b.up)

c.up = r_prod_up (a.up, b.up).

(8) a.lo ≤ 0 ≤ a.up, b.lo ≤ b.up ≤ 0

Then

a.up × b.lo ≤ a.up × b.up ≤ 0 ≤ a.lo × b.up ≤ a.lo × b.lo

so we put

c.lo = r_prod_down(a.up, b.lo)

c.up = r_prod_up (a.lo, b.lo).

(9) a.lo ≤ 0 ≤ a.up, b.lo ≤ 0 ≤ b.up

Then

a.lo × b.up, a.up × b.lo $\leq$ 0

a.lo × b.lo, a.up × b.up $\geq$ 0

so we put

c.lo = min(r_prod_down(a.lo, b.up), r_prod_down(a.up, b.lo))

c.up = max(r_prod_up (a.lo, b.lo), r_prod_up (a.up, b.up))

11. interval i_inv(a)

interval a;

This routine returns an interval b containing $1 +_i a$, provided $0 \notin a$. The routine checks to see whether 0 is in a. If it is not then it returns b, defined by:

b.lo = r_inv_down(a.up) $\leq 1/x$ for all $x \in a$

b.up = r_inv_up (a.lo) $\geq 1/x$ for all $x \in a$.

Then b contains the interval $1 +_i a$.

12. interval i_quot(a,b)

interval a,b;

This routine returns an interval containing $a +_i b$, for $0 \notin a$. It is defined simply in terms of i_prod and i_inv as i_prod(a, i_inv(b)).

13. interval i_power(a,k)

interval a; int k;

This routines returns an interval b containing a^k, for $k \geq 0$, and for a contained within the non-negative reals. It performs a check on a and k. If all is well, it returns b, defined by:

b.lo = r_power_down(a.lo, k) $\leq x^k$ for all $x \in a$

b.up = r_power_up (a.up, k) $\geq x^k$ for all $x \in a$.

Thus a^k is contained in b.

14. interval i_abs(a)

    interval a;

This routine returns an interval b containing the set of absolute values of a. The precise definition of b is:

    If a is contained within the non-negative reals, then b = a

    If a is contained within the non-positive reals, then b = -a

    Otherwise we have a.lo < 0 < a.up, and then

    b = [0, max{a.up, -a.lo}].

With this definition {| x | : x ∈ a} is contained in b.

15. double r_abs(a)

    interval a;

This routine returns an upper bound on the absolute value of elements of a. This upper bound may be defined simply as max{-a.lo, a.up}.

16. int int_zero(a)

    interval a;

This routine returns 1 if 0 ∈ a and 0 otherwise.

The following are a few routines on interval matrices and vectors. These routines i_matmult, i_matvec, r_matcol, i_matmult4, i_matvec4, r_matcol4, i_det4 are straightforward implementations of standard algorithms using the interval routines defined above. The definitions are specific to the requirements of the program circ_proof.

17. i_matmult(a,b,c,m)

    interval a[164][164], b[164][164], c[164][164]; int m;

This routine returns in the interval matrix c, the product $a \times_i b$. The

rows and columns of a, b, c range from 0 to m. The routine copies each column of b into an interval vector for increased efficiency (C stores arrays in rows, so that moving down columns is time consuming).

18. i_matvec(a,b,c,m)

    interval a[164][164], b[164], c[164]; int m;

This routine returns in the interval vector c, the product of a x_i b. The rows and columns of a, and the rows of b, c all range from 0 to m.

19. double r_matcol(a,m1,m2,m3,m4)

    interval a[164][164]; int m1, m2, m3, m4;

This routine returns the maximum of the L^1-norms of the columns of the submatrix of a[i][j] with m1 $\leq$ j $\leq$ m2, m3 $\leq$ i $\leq$ m4.

20. i_matmult4(a,b,c,m)

    interval a[4][4], b[4][4], c[4][4]; int m;

    i_matvec4(a,b,c,m)

    interval a[4][4], b[4], c[4]; int m;

    double r_matcol4(a,m1,m2,m3,m4)

    interval a[4][4]; int m1, m2, m3, m4;

These routines are entirely analagous to the routines in 17 - 19 but for 4 x 4 matrices instead of 164 x 164 matrices.

21. interval i_det4(a)

    interval a[4][4];

This routine returns an interval containing the determinant of the 4 x 4 interval matrix a. This is calculated using the cofactor method

expanding on the last column of a and its submatrices. The method has been chosen because it is simple to implement and because it involves a a great deal of factoring (useful for reducing error). In fact we expand on the last column because (in our application) the intervals in this column are wide (especially a[3][3]) and it is important to factor out such intervals. The actual definition of i_det4 is:

$$\sum_{i=0}^{3} (-1)^i \, a[i][3] \quad \times_i \quad \sum_{j=0,\,j\neq i}^{3} (-1)^j \, a[j][2]$$

$$\times_i \quad \sum_{k=0,\,k\neq i,j}^{3} (-1)^k \, a[k][1] \quad \times_i \quad \sum_{\ell=0,\,\ell\neq i,j,k}^{3} a[\ell][0]$$

where the summation signs mean interval sums.

A8.3 Function Ball/Vector Operations

Function balls/vectors are implemented in C as the following structure:

```
typedef struct {
        interval p[81];  /* polynomial part              */
        double   h;      /* high order function norm */
        double   e;      /* error function norm      */
        interval a;      /* centre of domain         */
        interval r;      /* radius of domain         */
} interval_function;
```

Let n denote the degree of the polynomial part, $n \leq 80$. Then if f is declared as an interval_function, then, in the notation of Chapter 4, f represents a function ball/vector with

v_i = f.p[i], i = 0, ..., n

v_h = f.h

v_e = f.e

The domain of the functions in this ball is any disc D(a, r), for which $a \in$ f.a and $r \in$ f.r. We shall have f.a.lo = f.a.up and f.r.lo = f.r.up, but this need not be the case.

The following routines are implemented on the computer: i_fzero, i_fconv, i_fnorm, r_ifnorm, i_fsmult, i_fscale, i_fsum, i_fdiff, i_fmult, i_fmult2, i_eval, i_fcomp, i_fcompl, i_fdcomp, i_fdcompl, i_fdkeval.

These routines are contained in the file i_function.c. The interval_function typedef statement and the declarations of the routines are contained in i_function.h.

We shall now describe the routines. We shall use freely the standard properties of the L^1-norm (Proposition A3.2). We make the following conventions:

(a) n, the degree of the polynomial part is fixed. For the routines in

i_function.c n is an external variable to be declared in the main program.

(b) All summations range from 0 to n unless indicated otherwise.

(c) All norms are the L^1-norms of Appendix 3.

(d) If f is declared to be an interval_function variable, then by f(x) we shall mean any function of the form:

$$f(x) = \Sigma\ f_i.y^i + f_h(y) + f_e(y)$$

where $f_i \in$ f.p[i], i = 0, .., n, $||f_h|| \leq$ f.h, $||f_e|| \leq$ f.e, $f_h(y) = O(y^{n+1})$ and $y = (x - a)/r$, for some $a \in$ f.a, $r \in$ f.r. Similar conventions will apply to other interval_function variables.

(e) When we write an interval operation we shall mean the computer implementation that is described in section A8.1 and A8.2.

1. interval_function i_fzero(a,r)

   interval a, r;

This routine returns the zero ball with f.a = a, f.r = r.

2. interval_function i_fconv(f)

   real_function f;

This routine returns an interval_function g that is a ball consisting of a single function f, defined on the domain f.a, f.r. (For a definition of a real_function data structure, see section A7.2 3) The routine simply converts the coefficients of the polynomial part of f to intervals consisting of a single point. The centre and radius of the domain of f, f.a and f.r, are converted to single point intervals g.a and g.r.

3. interval i_fnorm(f)

   interval_function f;

This routine returns an interval s that bounds the L^1-norms of the functions in f. We have the following estimate

$$\Sigma\ |f_i| + \|f_h\| - \|f_e\| \leq \|f\| \leq \Sigma\ |f_i| + \|f_h\| + \|f_e\|$$

The reason for this estimate is that while the polynomial part and the high order function contain terms of different orders, and therefore their norms add, the error function contains terms of all orders which may either increase or decrease the norm of f by a maximum of $\|f_e\|$. We note that

$$0 \leq \|f_h\| \leq f.h, \quad 0 \leq \|f_e\| \leq f.e$$

and so

$$\Sigma\ |f_i| - f.e \leq \|f\| \leq \Sigma\ |f_i| + f.h + f.e$$

These bounds are obtained by adding together the intervals i_abs(f.p[i]), i = 0, ..., n (which contain bounds on the absolute values of the elements in the intervals f.p[i], i = 0, ..., n) together with the intervals [0, f.h] and [-f.e, f.e].

4. double r_ifnorm(f)

interval_function f;

This routine returns an upper bound on the L^1-norms of the functions in f. The routine is defined simply as r_abs(i_fnorm(f)).

5. interval_function i_fsmult(a,f)

interval a; interval_function f;

This routine returns an interval function g that corresponds to multiplying functions in f by "scalar" real numbers in a. We note that for $a \in R$,

$$a(\Sigma f_i.y^i + f_h(y) + f_e(y)) = \Sigma\ (af_i).y^i + a.f_h(y) + a.f_e(y)$$

and

$$\|a.f_h\| \leq |a|.\|f_h\|, \quad \|a.f_e\| \leq |a|.\|f_e\|.$$

$a.f_h(y)$ comprises terms that are $O(y^{n+1})$. A precise definition of g is:

g.p[i] = a $\times_i$ f.p[i], i = 1, ..., n

g.h = r_prod_up(r_abs(a), f.h)

g.e = r_prod_up(r_abs(a), f.e)

g.a = f.a, g.r = f.r.

6. interval_function i_fscale(f,a,r)

interval_function f; interval a,r;

This routine returns an interval function g scaled to a domain with centre in a and radius in r i.e. g = (f - a)/r. We note that for a, r $\in$ R,

$$(\Sigma f_i.y^i + (y) + f_e(y) - a)/r = (f_0 - a)/r + \Sigma f_i/r.y^i + 1/r.f_h(y) + 1/r.f_e(y)$$

(where the second summation ranges for i = 1 to n) and that

$\|1/r.f_h\| \leq 1/|r|.\|f_h\| \leq 1/|r|.f.h$

$\|1/r.f_e\| \leq 1/|r|.\|f_e\| \leq 1/|r|.f.e.$

$1/r.f_h(y)$ comprises only terms that are $O(y^{n+1})$. A precise definition of g is:

ri = i_inv(r)

g.p[0] = (f.p[0] $-_i$ a)$*_i$ri

g.p[i] = f.p[i] $\times_i$ ri , i = 1, ..., n.

g.h = r_prod_up(r_abs(ri), f.h)

g.e = r_prod_up(r_abs(ri), f.e)

g.a = f.a, g.r = f.r.

7. interval_function i_fsum(f,g)

inteval_function f,g;

This routine returns an interval_function h corresponding to the sum of two functions in f and g i.e. h = f + g. The domain of f and g are

assumed to be the same (the routine does not check that this is indeed so). We have the following:

$$(\Sigma f_i.y^i + f_h(y) + f_e(y)) + (\Sigma g_i.y^i + g_h(y) + g_e(y)) =$$

$$\Sigma(f_i + g_i).y^i + (f_h(y) + g_h(y)) + (f_e(y) + g_e(y)).$$

and

$$\|f_h + g_h\| \leq \|f_h\| + \|g_h\| \leq r\_sum\_up(f.h, g.h)$$

$$\|f_e + g_e\| \leq \|f_e\| + \|g_e\| \leq r\_sum\_up(f.e, g.e).$$

The function $(f_h(y) + g_h(y))$ comprises only terms that are $O(y^{n+1})$.

The precise definition of h is:

$h.p[i] = f.p[i] +_i g.p[i]$, $i = 0, \ldots, n$

h.h = r_sum_up(f.h, g.h)

h.e = r_sum_up(f.e, g.e)

h.a = f.a, h.r = f.r

8. interval_function i_fdiff(f,g)

interval_unction f,g;

This routine returns an interval_function h corresponding to the difference of two functions in f and g i.e. h = f - g. The domain of f and g are assumed to be the same, but again there is no check. We have the following:

$$(\Sigma f_i.y^i + f_h(y) + f_e(y)) - (\Sigma g_i.y^i + g_h(y) + g_e(y)) =$$

$$\Sigma(f_i - g_i).y^i + (f_h(y) - g_h(y)) + (f_e(y) - g_e(y)).$$

and

$$\|f_h - g_h\| \leq \|f_h\| + \|g_h\| \leq r\_sum\_up(f.h, g.h)$$

$$\|f_e - g_e\| \leq \|f_e\| + \|g_e\| \leq r\_sum\_up(f.e, g.e).$$

The function $(f_h(y) - g_h(y))$ comprises only terms that are $O(y^{n+1})$.

The precise definition of h is:

$h.p[i] = f.p[i] -_i g.p[i]$, $i = 0, \ldots, n$

h.h = r_sum_up(f.h, g.h)

```
h.e    = r_sum_up(f.e, g.e)
h.a = f.a, h.r = f.r
```

9. `interval_function i_fmult(f,g)`

```
interval_function f,g;
```

This routine returns an interval_function h corresponding to the product of functions in f and g. i.e. h = f × g. The domain of f and g are assumed to be the same but there is no check. We have the following:

$$(\Sigma f_i.y^i + f_h(y) + f_e(y))\times(\Sigma g_i.y^i + g_h(y) + g_e(y))$$

$$= (\Sigma f_i.y^i).(\Sigma g_i.y^i) + (\Sigma f_i.y^i).g_h(y) + (\Sigma f_i.y^i).g_e(y)$$

$$+ f_h(y).(\Sigma g_i.y^i) + f_h(y).g_h(y) + f_h(y).g_e(y) + f_e(y).(\Sigma g_i.y^i)$$

$$+ f_e(y).g_h(y) + f_e(y).g_e(y))$$

The first term of the right hand side of this equation can be written:

$$\sum_{i=0}^{n} \left(\sum_{j=0}^{i} f_j.g_{i-j}\right).y^i \quad + \quad \sum_{i=n+1}^{2n} \left(\sum_{j=i-n}^{n} f_j.g_{i-j}\right).y^i$$

The first of these expressions consists of terms of degree $\leq n$. The second consists of terms of degree $\geq n+1$. Thus we may write the product of f and g as $h_p + h_h + h_e$ where:

$$h_p(y) = \sum_{i=0}^{n} \left(\sum_{j=0}^{i} f_j.g_{i-j}\right).y^i$$

$$h_h(y) = \sum_{i=n+1}^{2n} \left(\sum_{j=i-n}^{n} f_j.g_{i-j}\right).y^i + (\Sigma f_i.y^i).g_h(y) + f_h(y).(\Sigma g_i.y^i)$$

$$+ f_h(y).g_h(y) + f_h(y).g_e(y) + f_e(y).g_h(y)$$

$$h_e(y) = (\Sigma f_i.y^i)g_e(y) + f_e(y).(\Sigma g_i.y^i) + f_e.g_e .$$

We note that all the terms in $h_h(y)$ are $O(y^{n+1})$. We have the following estimates:

$$\|h_h\| \leq \left\| \sum_{i=n+1}^{2n} \left(\sum_{j=i-n}^{n} f_j . g_{i-j} \right) . y^i \right\| + (\Sigma|f_i|).\|g_h\| + \|f_h\|.(\Sigma|g_i|)$$

$$+ \|f_h\|.\|g_h\| + \|f_h\|.\|g_e\| + \|f_e\|.\|g_h\|$$

$$\|h_e\| \leq (\Sigma|f_i|).\|g_e\| + \|f_e\|.(\Sigma|g_i|) + \|f_e\|.\|g_e\|$$

Upper bounds for these quantities are calculated from the bounds f.h, f.e, g.h, g.e. The precise definition of h is:

$$h.p[i] = \sum_{j=0}^{i} f.p[j] \times_i g.p[i-j] \text{ for } i = 0, \ldots, n.$$

(The summation sign refers to interval summation.)

The upper bound h.h is an upper bound (obtained by using the routine r_sum_up) for the sum of the following:

aa, ab × g.h, f.h × ac, f.e × g.h, f.h × g.e, f.h ×g.h

Here aa is an upper bound for

$$\left\| \sum_{i=n+1}^{2n} \left(\sum_{j=i-n}^{n} f_j . g_{i-j} \right) . y^i \right\|$$

obtained by evaluating the coefficient

$$\left(\sum_{j=i-n}^{n} f.p[j] \times g.p[i-j] \right) \quad \text{for } i = n+1, \ldots, 2n.$$

ab and ac are upper bounds for $\Sigma |f_i|$ and $\Sigma |g_i|$ respectively.

Similarly, h.e is an upper bound for the sum of:

ab × g.e, f.e × ac, f.e × g.e.

Finally, h.a = f.a, h.r = f.r.

10. interval_function i_fmult2(f,g)

interval_function f,g;

This routine returns an interval_function h corresponding to the product of two functions in f, g, i.e. h = f × g, with assumption that all functions in g are at most linear. The domain of f and g are assumed to be equal but there is no check. Note that, by assumption,

g.p[i] = [0,0] for i = 2, ..., n

g.h = g.e = 0

We have the following:

$$(\Sigma f_i.y^i + f_h(y) + f_e(y)) \times (g_0 + g_1.y)$$

$$= f_0.g_0 + \sum_{i=1}^{n}(f_i.g_0 + f_{i-1}.g_1).y^i + (f_n.g_1)y^{n+1}$$

$$+ f_h(y).(g_0 + g_1.y) + f_e(y).(g_0 + g_1.y).$$

Writing $h = h_p + h_h + h_e$, as before, we have:

$$h_p(y) = f_0.g_0 + \sum_{i=1}^{n}(f_i.g_0 + f_{i-1}.g_1).y^i$$

$$h_h(y) = f_n.g_1.y^{n+1} + f_h.(g_0 + g_1.y)$$

$$h_e(y) = f_e.(g_0 + g_1.y)$$

We note that $h_h(y)$ comprises only terms that are $O(y^{n+1})$. A precise definition of the interval_function h is:

$$h.p[0] = f.p[0] \times_i g.p[0]$$

$$h.p[i] = f.p[i] \times_i g.p[0] + f.p[i-1] \times_i g.p[1],\ i = 1, \ldots, n$$

h.h is an upper bound (obtained using the r_sum_up routines) for the sum of the two quantities

$$|f_n|.|g_1|,\ f.h \times (|g_0| + |g_1|)$$

while h.e is an upper bound for:

$$f.e \times (|g_0| + |g_1|).$$

$h.a = f.a,\ h.r = f.r$

Before moving on to the next routine, we make some general remarks about the evaluation and composition routines.

(a) The routines are all "linear in the first argument." This means that one can treat the three different parts of the function ball/vector i.e. the polynomial part, the high order function and the error function, as independent of each other, and then add up the results at the end.

(b) We deal with the polynomial part first. This usually is completely straightforward and merely requires implementation of a standard

algorithm.

(c) Whenever the polynomial part of a function ball is evaluated (either at a interval or at a interval_function (when composing two functions)) then the following method of evaluating polynomials is used:

$$(\ldots((f_n.x + f_{n-1}).x + f_{n-2}).x + \ldots. f_1).x + f_0 \qquad (A8.1)$$

(d) The treatment of the high order and error functions are often similar, especially when dealing with differentiation.

(e) When considering the high order and error functions we shall write

$$f_h(y) = \sum_{i=n+1}^{\infty} d_i.y^i \quad \text{and} \quad f_e(y) = \sum_{i=0}^{\infty} e_i.y^i.$$

11. interval i_eval(f,a)

interval_function f; interval a;

This routine returns an interval c that bounds the value of a function in f at any real value in the interval a i.e. c = f(a). The routine checks that all elements of a are contained within the domain of definition of any function in f. The routine first scales a to the domain of f i.e. sets an interval b = (a - f.a)/f.r. The polynomial part is evaluated using the algorithm (A8.1). To this is added intervals that bound $f_h(b)$ and $f_e(b)$. We have the following estimates. Let $b \in D(0,1)$.

$$|f_h(b)| \leq |b|^{n+1}.\|f_h\| \quad \text{and} \quad |f_e(b)| \leq \|f_e\|.$$

Thus intervals bounding $f_h(b)$ and $f_e(b)$ are:

$[-|b|^{n+1}.f.h,\ |b|^{n+1}.f.h]$ and $[-f.e, f.e]$ respectively.

The first of these intervals is evaluated using the routine r_power_up.

12. interval_function i_fcomp(f,g)

    interval_function f,g;

This routine returns an interval_function h corresponding to the composition of functions in f with those in g. i.e. h = f o g. The function g is scaled to the domain of f, using the routine i_fscale. This scaled interval_function is denoted by q. We consider the polynomial part, high order function and the error function of f separately.

(a) polynomial part, f_p.

f_p o q is evaluated directly using the algorithm (A8.1) and the routines i_fmult and i_fsum.

(b) high order function f_h.

It is simple to get a bound for f_h o q viz. $\|q\|^{n+1}.\|f_h\|$. However, in general, q will have a non-zero constant term and f_h o q will contain terms of all orders. It must therefore be added to the error function of the function h. However, as in Eckmann et al (1982), it is possible to divide f_h o q into a high order part and an error part as follows: Consider a term of f_h o q which has the form $d_i.q^i$, $i \geq n+1$. We write

$$d_i.q^i = d_i.q^{i-n-1}(q - q(0))^{n+1} + d_i.q^{i-n-1}(q^{n+1} - (q - q(0)^{n+1})$$

The first term is $O(y^{n+1})$ while the second term may be of any order. We include the first term in the function h_h while the second belongs to the error function h_e. Summing over $i \geq n+1$, we get the following bound for h.h and h.e from the function f_h.

$$\|h_h\| = \sum_{i=n+1}^{\infty} |d_i|.\|q^{i-n-1}\|.\|(q - q(0)\|^{n+1} \leq f.h \times \|q - q(0)\|^{n+1}$$

$$\|h_e\| = \sum_{i=n+1}^{\infty} |d_i|.\|q^{i-n-1}\|.\|q^{n+1} - (q - q(0))^{n+1}\|$$

$$\leq f.h \times \|q^{n+1} - (q - q(0))^{n+1}\|$$

We claim that:

$$\|q^{n+1} - (q - q(0))^{n+1}\| \leq \|q\|^{n+1} - \|q - q(0)\|^{n+1}.$$

Write $\alpha = q(0)$, $\beta = q - q(0)$. Then $q = \beta + \alpha$ and

$$\|(\beta + \alpha)^{n+1} - \beta^{n+1}\| \leq \sum_{i=1}^{n+1} C_i^{n+1} |\alpha|^i . \|\beta\|^{n-i+1}$$

$$\leq (\|\beta\| + |\alpha|)^{n+1} - \|\beta\|^{n+1} = \|\beta + \alpha\|^{n+1} - \|\beta\|^{n+1}$$

$$= \|q\|^{n+1} - \|q - q(0)\|^{n+1}$$

for $\|\beta\| + |\alpha| = \|\beta + \alpha\|$ as β has no constant term. The following method is used to evaluate the norms:

We set s = i_fnorm(q), so that $\|q\| \in s$

and then set q.p[0] = [0,0] and evaluate sl = i_fnorm(q)

so that $\|q - q(0)\| \in sl$. We write sn = i_power(s,n+1),

sln = i_power(sl, n+1), so $\|q\|^{n+1} \in sn$ and $\|q - q(0)\|^{n+1} \in sln$

and $\|q\|^n - \|q - q(0)\|^{n+1} \in$ i_diff(sn, sln).

(c) Error function, f_e. There is a simple bound for $f_e \circ q$:

$$\|f_e \circ q\| \leq \sum_{i=0}^{\infty} |e_i| . \|q^i\| \leq \|f_e\| \leq f.e$$

This function is added to the error function of h so that h.e is increased by f.e.

13. interval_function i_fcompl(f,g)

interval_function f,g

This routine returns an interval_function h corresponding to the composition of functions in f with those in g (i.e. h = f o g), with assumption that all functions in g are at most linear. The interval_function g is scaled to the domain of f, using the routine i_fscale. This routine is identical to the routine i_fcomp except that the routine i_fmult2 is used instead of i_fmult in the evaluation of the polynomial part of f.

The last three routine are differentiation followed by composition/evaluation routines. We make the following points:

(a) They each require an estimate of the following form:

Let $k \geq 1$ be an integer. Let $0 \leq t < 1$. Then

$$\sup \{ (i + k)(i + k - 1)\ldots(i + 1).t^i : i \geq 0\}$$

is equal to $(m_1 + k)\ldots(m_1 + 1).t^{m_1}$, where m_1 satisfies

$$m_1 < k.t/(1 - t) \leq m_1 \qquad \text{(A8.2)}$$

This can be easily seen from the following:

$$(i + k)\ldots(i + 1)t^i/((i + k - 1)\ldots it^{i-1}) =$$

$$(i + k)t/i \leq 1 \text{ iff } i \geq k.t/(1 - t)$$

Thus the maximum value is attained for $i = m_1$ given above.

(b) Since we are differentiating with respect to x, and the functions f_p, f_h, f_e are all expressed as functions of the scaled variable $y = (x - f.a)/f.r$, we must take this into account. Since $d^k/dx^k \equiv d^k/dy^k.1/f.r^k$, this is simply a matter of dividing the result by $f.r^k$. This if done in the case of the composition routines by means of the routine i_fscale, with a = [0,0].

14. interval_function i_fdcomp(f,g)

    interval_function f,g;

This routine returns an interval_function h corresponding to the differentiation of the functions in f followed by composition with those in g i.e h = Df o g. The interval_function g is scaled to the domain of f, using the routine i_fscale. The scaled interval_function is denoted by q. We again consider the polynomial part, the high order function and the error function of f separately.

(a) Polynomial part, f_p.

This is computed directly by forming an interval_function df that contains the derivative of all the functions in the polynomial part f_p. This is simply:

df.p[i-1] = [i,i]*f.p[i], i = 1, ..., n, df.p[n] = [0,0]

df.h = df.e = 0.0, df.a = f.a, df.r = df.r.

The routine i_fcomp is used to obtain Df_p o q.

(b) High order function f_h and error function f_e.

These two cases are similar, so we consider them together. We have the following estimates:

$$\|Df_h \text{ o } q\| \leqslant \sum_{i=n+1}^{\infty} i.|d_i|.\|q^{i-1}\|) \leqslant \|f_h\|.\ \sup\{(i+1).\|q\|^i : i \geqslant n\}$$

$$\|Df_e \text{ o } q\| \leqslant \sum_{i=1}^{\infty} i.|e_i|.\|q^{i-1}\|) \leqslant \|f_e\|.\ \sup\{(i+1).\|q\|^i : i \geqslant 0\} \quad \text{(A8.3)}$$

These are bounded using the above remarks. Note that if m_1 in (A8.2) is less than n, then this first bound is:

$$\|Df_h \text{ o } q\| \leqslant \|f_h\|.(n+1).\|q\|^n.$$

The routine finds upper and lower bounds for t/(1 - t), where $t \geqslant \|q\|$ and hence finds the integer m_1 satsifying (A8.2) with k = 1. The first of the suprema in (A8.3) above is denoted by v and the second by u. The bounds f.e x u and f.h x v are added to the error function bound h.e obtained from the polynomial part f_p. The whole interval_function is then divided by f.r.

15. interval_function i_fdcompl(f,g)

    interval_function f,g;

This routine returns an interval_function h corresonding to the differentiation of the functions in f followed by composition with those in g (i.e h = Df o g), with assumption that all functions in g are at most linear. The interval_function g is scaled to the domain of f, using the routine i_fscale. This routine is identical to the routine i_dfcomp except that the routine i_fcompl is used instead of i_fcomp in the evaluation of the polynomial part of f.

16. interval i_fdkeval(f,k,a)

    interval_function f; int k; interval a;

This routine returns an interval which contains the result of differentiating f k times (k $\geq$ 1) and evaluating at a. We consider the polynomial part of f first.

(a) Polynomial part, f_p. The kth derivative of the polynomial part $D^k f_p$ is placed in the interval_function df. The coefficients of df are as follows:

df.p[i] = fac[i] $\times$ f.p[i+k] for i = 0, ..., n - k

df.p[i] = [0, 0] for i = n - k + 1, ..., n

df.h = df.e = 0, df.a = f.a, df.r = f.r

where fac[i] = (i + k)....(i + 1) for i = 0,..., n - k + 1

(fac[i] is calculated at i = n - k + 1 for use in the calculation of the bounds in the high order part). The routine i_eval is used to evaluate this interval_function df at a.

(b) High order function f_h, and error function f_e. We have the following estimates: Writing y for (a - f.a)/f.r

$$|D^k f_h(y)| \leq \sum_{i=n+1-k}^{\infty} (i+k)\ldots(i+1).|d_{i+k}|.|y^i|$$

$$\leq \|f_h\|.\sup\{(i+k)\ldots(i+1).|y|^i : i \geq n+1-k\} \qquad \text{(A8.4)}$$

$$|D^k f_e(y)| \leq \sum_{i=0}^{\infty} (i+k)\ldots(i+1)|e_{i+k}|.|y^i|$$

$$\leq \|f_e\|.\sup\{(i+k)\ldots(i+1).|y|^i : i \geq 0\}$$

These quantities are bounded using the remarks above. Note that if m1 in (A8.2) is less than n - k + 1, then this first bound is

$$|D^k f_h(y)| \leq \|f_h\|.(n+1).....(n-k+2)|y|^{n-k+1}.$$

The routine finds upper and lower bounds for k.t/(1 - t), where t $\geq$ |y| and finds the integer m_1 satisfying (A8.2). The first of the suprema in (A8.4) above is denoted by v and the second by u. An interval [-w, w] where w $\geq$ f.a $\times$ u + f.h $\times$ v is added to the interval obtained from the polynomial part f_p. The whole interval is then divided by $f.r^k$.