Journal of Statistical Physics, Vol. 46, Nos. 3/4, 1987

A Complete Proof of the Feigenbaum Conjectures

Jean-Pierre Eckmann[1] **and Peter Wittwer**[2]

Received October 31, 1986

The Feigenbaum phenomenon is studied by analyzing an extended renormalization group map $\mathscr{M}$. This map acts on functions Φ that are jointly analytic in a "position variable" (t) and in the parameter (μ) that controls the period doubling phenomenon. A fixed point Φ^* for this map is found. The usual renormalization group doubling operator $\mathscr{N}$ acts on this function Φ^* simply by multiplication of μ with the universal Feigenbaum ratio $\delta^* = 4.669201...$, i.e., $(\mathscr{N}\Phi^*)(\mu, t) = \Phi^*(\delta^*\mu, t)$. Therefore, the one-parameter family of functions Ψ^*_μ, $\Psi^*_\mu(t) = \Phi^*(\mu, t)$, is invariant under $\mathscr{N}$. In particular, the function Ψ^*_0 is the Feigenbaum fixed point of $\mathscr{N}$, while Ψ^*_μ represents the unstable manifold of $\mathscr{N}$. It is proven that this unstable manifold crosses the manifold of functions with superstable period two transversally.

KEY WORDS: Nonlinear functional equation; renormalization group; Feigenbaum phenomenon; computer-assisted proof; rigorous bounds on critical indices.

1. DEFINITION OF THE OPERATOR $\mathscr{M}$

We present in the following a new and more complete proof of the Feigenbaum conjectures[1,2,6] similar to the ideas of Vul and Khanin[3] (see also Vul *et al.*[4])

The problem we solve can be formulated as follows: consider the set $\mathscr{A}_0$ of functions of two complex variables μ and t that are analytic in the domain $D \subset \mathbb{C}^2$. Here,

$$D = \{|\mu| < 1\} \times \{|t - 1| < \rho\}$$

[1] Université de Genève, Physique Théorique, 1211 Geneva 4, Switzerland.

[2] Rutgers University, Department of Mathematics, New Brunswick, New Jersey 08903.

with $\rho = 2$. We next define $\mathscr{A}$ as the subset of those $\Phi \in \mathscr{A}_0$ that take real values for real arguments μ and t, and satisfy

$$\begin{aligned} \Phi(\mu, 0) &= 1 \\ \Phi(0, 1) &= \bar{\lambda} \\ \partial_\mu \Phi(0, 1) &= \bar{\varepsilon} \end{aligned} \tag{1.1}$$

where $\bar{\lambda} = -1/2.5029078750957$ and $\bar{\varepsilon} = 0.0005$. As we shall see, these three equations fix the scale and the origin of the first variable μ and the scale of the second variable t. We next choose a constant $\bar{\sigma} = 0.23$ and define the operator $\mathscr{M}$ by the following prescriptions:

(a) Set $\lambda(\mu) = \Phi(\bar{\sigma}\mu, 1)$.

(b) Define $\mathscr{M}_0 \Phi(\mu, t) = \lambda(\mu)^{-1}\, \Phi(\bar{\sigma}\mu, (\Phi(\bar{\sigma}\mu, \lambda^2(\mu) t))^2)$.

(c) Determine μ_0 as the solution of $(\mathscr{M}_0 \Phi)(\mu_0, 1) = \bar{\lambda}$.

(d) Define $\tau = \partial_\mu (\mathscr{M}_0 \Phi)(\mu_0, 1)$.

(e) Set $(\mathscr{M}\Phi)(\mu, t) = (\mathscr{M}_0 \Phi)(\bar{\varepsilon}\mu/\tau + \mu_0, t)$.

It is easy to see that if $\mathscr{M}$ is defined, then $\mathscr{M}\Phi$ satisfies the normalizations (1.1) of the set $\mathscr{A}$.

The choice of $\bar{\sigma}$ will guarantee that $\mathscr{M}_0 \Phi$ is analytic in the same domain D as Φ, which is convenient for the use of the computer. (If $\bar{\sigma}$ were equal to 1, we would expect the action of $\mathscr{M}_0$ to be dilation of the μ argument by $\delta^* \approx 4.66$. Therefore it is adequate to choose $\bar{\sigma} \approx 1/\delta^*$.)

Below, we will equip a subset of $\mathscr{A}$ with a norm. We then use the computer to show that $\mathscr{M}$ is defined and is a contraction of a suitable ball in this subset of $\mathscr{A}$. This will allow us to conclude that $\mathscr{M}$ has a fixed point.

2. OUTLINE OF THE PROOF

We now present the approach in more detail. We want to work with the space $\mathscr{L}$ of functions that are analytic on the product of two unit disks, equipped with the norm

$$\|F\| = \sum_{i,j} |f_{ij}|$$

where $F(\mu, t) = \sum_{ij} f_{ij} \mu^i t^j$. In view of the definition of $\mathscr{A}$, we write Φ as

$$\Phi(\mu, t) = H\left(\mu, \frac{t-1}{\rho}\right) = 1 + tF\left(\mu, \frac{t-1}{\rho}\right) \tag{2.1}$$

with

$$F(\mu, t) = \bar{\lambda} - 1 + \bar{\varepsilon}\mu + At + G(\mu, t)$$

where $G(\mu, t)$ is of second order in μ and t, and A is a real number. This choice of coordinates ensures the correct normalizations. Recall that $\rho = 2$.

We discuss now briefly the various spaces introduced so far. The functions F span a hyperplane $\mathscr{L}'$ of $\mathscr{L}$ of codimension 2. The map $\mathscr{R}$ defined below will map $\mathscr{L}'$ to itself. The hyperplane $\mathscr{L}'$ is a translate (by $\bar{\lambda} - 1 + \bar{\varepsilon}\mu$) of a linear subspace $\mathscr{L}''$ of $\mathscr{L}$ (again of codimension 2), equipped with the same norm as $\mathscr{L}$, and $D\mathscr{R}$ will map $\mathscr{L}''$ to itself. The norm of $\mathscr{L}$ induces by (2.1) a natural norm on $\mathscr{A}$, and we equip $\mathscr{A}$ with this norm.

The prescriptions (a)–(e) of Section 1 induce on $\mathscr{L}'$ a map from F to a "new" F, called $\mathscr{R}F$, which we describe now:

(a) Compute $H(\mu, t) = 1 + (\rho t + 1)\, F(\mu, t)$.

(b) Set $\lambda(\mu) = H(\bar{\sigma}\mu, 0)$.

(c) Define

$$F_1(\mu, t) = F\left(\bar{\sigma}\mu, \lambda^2(\mu)t + \frac{\lambda^2(\mu) - 1}{\rho}\right)$$

(d) Define $F_2(\mu, t) = \lambda^2(\mu)\, F_1(\mu, t)$.

(e) Define $F_3(\mu, t) = 2F_2(\mu, t) + (\rho t + 1)\, F_2(\mu, t)^2$.

(f) Define $H_3(\mu, t) = 1 + (\rho t + 1)\, F_3(\mu, t)$.

(g) Finally,

$$H_4(\mu, t) = \frac{1}{\lambda(\mu)} H\left(\bar{\sigma}\mu, \frac{1}{\rho}\,[H_3(\mu, t) - 1]\right)$$

(h) Now look for a μ_0 solving $H_4(\mu_0, 0) = \bar{\lambda}$ and define $\tau = \partial_\mu H_4(\mu_0, 0)$.

(i) Define $H_5(\mu, t) = H_4(\bar{\varepsilon}\mu/\tau + \mu_0, t)$.

(j) $\mathscr{R}$ is finally given by $\mathscr{R}F(\mu, t) = [H_5(\mu, t) - 1]/(\rho t + 1)$.

We construct (using a program for nonrigorous calculations) a polynomial F_0 of degree 16, and we verify with a program using rigorous error bounds that

$$\|\mathscr{R}F_0 - F_0\| < \varepsilon = 1.78 \times 10^{-11}$$

We next check that $D\mathscr{R}$ is a contraction on a ball of radius $\beta = 2.1 \times 10^{-10}$, centered at F_0. For this, we write explicitly the tangent map $D\mathscr{R}_F$ (evaluated at F) acting on δF:

$$\delta\lambda(\mu) = \delta F(\bar{\sigma}\mu, 0)$$

$$\delta H(\mu, t) = (\rho t + 1)\, \delta F(\mu, t)$$

$$\begin{aligned}
\delta F_1(\mu, t) &= \delta F(\bar\sigma\mu, \lambda^2 t + \cdots) \\
&\quad + \partial_t F(\bar\sigma\mu, \lambda^2 t + \cdots)\cdot 2\lambda\cdot\delta\lambda\cdot(t+1/\rho) \\
\delta F_2(\lambda, t) &= \lambda[\delta F_1(\mu, t)\lambda + F_1(\mu, t)\, 2\delta\lambda] \\
\delta F_3(\mu, t) &= 2\delta F_2(\mu, t)[1 + (\rho t + 1)\, F_2(\mu, t)] \\
\delta H_3(\mu, t) &= (\rho t + 1)\, \delta F_3(\mu, t) \\
\delta H_4(\mu, t) &= \lambda^{-1}\{-\delta\lambda H_4 + \delta H(\bar\sigma\mu, [H_3(\mu, t) - 1]/\rho) \\
&\quad + \partial_t H(\bar\sigma\mu, [H_3(\mu, t) - 1]/\rho)\cdot \rho^{-1}\delta H_3(\mu, t)\} \\
\delta\mu_0 &= -\delta H_4(\mu_0, 0)/\tau \\
\delta\tau &= \partial_\mu\, \delta H_4(\mu_0, 0) + \partial_\mu^2 H_4(\mu_0, 0)\, \delta\mu_0 \\
\delta H_5(\mu, t) &= \delta H_4\left(\frac{\varepsilon}{\tau}\mu + \mu_0, t\right) + \left(\delta\mu_0 - \frac{\varepsilon}{\tau^2}\delta\tau\mu\right)\, \partial_\mu H_4\left(\frac{\varepsilon}{\tau}\mu + \mu_0, t\right) \\
(D\mathscr{R}_F\, \delta F)(\mu, t) &= \delta(\mathscr{R}F)(\mu, t) = \delta H_5(\mu, t)/(\rho t + 1)
\end{aligned}$$

We bound $D\mathscr{R}$ as a map from $\mathscr{L}''$ to itself, by performing the following calculation. For a finite number of polynomial basis vectors δF (namely $\mu^i t^j$, with $i + j \leqslant 16$ and $i + j \geqslant 2$ or $i = 0$ and $j = 1$), and for a "higher order term" the program computes the norm of the image under $D\mathscr{R}_{F_{0\beta}}$, where $F_{0\beta}$ is the ball of radius $\beta = 2.1\times 10^{-10}$ centered at F_0 mentioned above. The sup of these norms is a bound on the operator norm of $D\mathscr{R}_F$ for every $F \in F_{0\beta}$ and it turns out that this sup is bounded by $\sigma = 0.7645$. Since $\varepsilon/(1-\sigma) < \beta$, we see that $\mathscr{R}$ contracts the ball of radius β into itself and therefore has a unique fixed point in this ball. Thus we have proved the following theorem.

Theorem 2.1. The map $\mathscr{R}$ has a fixed point F^* in $\mathscr{L}'$, and $\|F^* - F_0\| < \varepsilon/(1-\sigma)$.

We denote by H^*, λ^*,... the quantities obtained by applying (a)–(j) to F^*. We then define the function Φ^* by the equation

$$\Phi^*(\mu, t) = 1 + tF^*\left(\mu_0^* + \mu, \frac{t-1}{\rho}\right)$$

If μ_0^* were equal to 0, then we would find $\Phi^* \in \mathscr{A}$. Since μ_0^* is not equal to 0, the domain of analyticity of Φ^* is not D, but

$$D^* = \{|\mu + \mu_0^*| < 1\} \times \{|t-1| < \rho\}$$

However, Φ^* still satisfies the normalizations (1.1) of the set $\mathscr{A}$. Theorem 2.1 implies the following theorem.

Theorem 2.2. The map $\mathscr{M}$ has a fixed point Φ^*.

We study next the doubling operator $\mathscr{N}$, which is defined by

$$(\mathscr{N}\Psi)(t) = \frac{1}{\lambda}\Psi([\Psi(\lambda^2 t)]^2)$$

where $\lambda = \Psi(1)$. The operator $\mathscr{N}$ acts on functions Ψ that are analytic in

$$I = \{t \in \mathbb{C} \mid |t-1| < \rho\}$$

and normalized to $\Psi(0) = 1$. Consider now the set of all functions Ψ that are analytic in I, are normalized to $\Psi(0) = 1$, and take real values for real arguments. We then call $\mathscr{H}$ the space we get when we equip these functions with the norm

$$\|\Psi\|_1 = \sum_{i=1}^{\infty} \frac{1}{\rho^i \cdot i!} |\partial_t^i \Psi|_{t=1}$$

By construction, the function $\Phi^*(0, \cdot)$ is a fixed point of $\mathscr{N}$ (this is the celebrated Feigenbaum function). We next define

$$\Psi_\mu^* = \Phi^*(\mu, \cdot)$$

By construction, $\Psi_\mu^* \in \mathscr{H}$. The above results imply the following proposition.

Proposition 2.3. The action of $\mathscr{N}$ on Ψ_μ^* is trivial, namely

$$\mathscr{N}\Psi_\mu^* = \Psi_{\delta^*\mu}^*$$

The constant $\delta^* = \bar{\varepsilon}/\partial_\mu H_4^*(\mu_0^*, 0)$ is called the "Feigenbaum constant" and occurs as a universal parameter in the theory of period-doubling bifurcations of one-parameter families of maps of the unit interval (see, e.g., Ref. 2). Our computer-assisted proof gives bounds on this constant δ^*:

$$\delta^* \in [4.66920159, 4.669201622]$$

Remark 2.4. The preceding proposition shows that Ψ_μ^* is the unstable manifold of $\mathscr{N}$ at its fixed point. In addition, we see that it is an analytic manifold. Also,

$$\partial_\mu \Psi_\mu^* |_{\mu=0}$$

is the eigenvector with eigenvalue δ^* of $D\mathscr{N}$ at Ψ_0^*.

We next indicate how we prove that the unstable manifold crosses transversally the manifold Σ formed by those functions of $\mathscr{H}$ that satisfy

$f(1)=0$, in addition to the normalization $f(0)=1$. We consider a short piece of the unstable manifold

$$\Psi_\mu^* \qquad \text{for} \quad \mu \in [0.3436, 0.4036]$$

We then verify with our program that $\mathcal{N}^4$ is defined on these functions and maps them to functions analytic on the domain I. We verify next that $\mathcal{N}$ is defined on $\mathcal{N}^4\Psi_\mu^*$, when we restrict the analyticity domain in t to $\{|t-1|<1.2\}$. We then check that

$$(\mathcal{N}^5\Psi_{\mu=0.4036}^*)(1)>0$$

and

$$(\mathcal{N}^5\Psi_{\mu=0.3436}^*)(1)<0$$

and furthermore

$$\partial_\mu(\mathcal{N}^5\Psi_\mu^*)(1)\neq 0 \qquad \text{for all} \quad \mu \in [0.3436, 0.4036]$$

This shows the following theorem.

Theorem 2.5. The unstable manifold of $\mathcal{N}$ crosses the surface Σ transversally.

Remark 2.6. By a similar calculation we show that four iterations of $\mathcal{N}$ map the local unstable manifold transversally across the surface Σ_M of "band merging functions," i.e., those functions satisfying $f(f(1)^2)=-f(1)$.

We next verify that $\mathcal{N}^5$ is defined on $\Psi_{\mu\in[0.09,0.43336]}^*$ (with the same restriction of domain for the fifth iteration as above). Since $\delta^*\cdot 0.09<0.4336$, this proves the following theorem.

Theorem 2.7. The local unstable manifold of $\mathcal{N}$ extends from Ψ_0^* to Σ.

Remark 2.8. We also verify that the local unstable manifold extends from Ψ_0^* to Σ_M.

3. THE COMPUTER-ASSISTED PROOF

Computer-assisted proofs have by now a certain tradition, going back to Lanford's seminal paper.[6] The general principles have been spelled out in detail in Refs. 7–9 (see also Ref. 12). The main ideas are as follows: On a computer, rigorous interval arithmetic is possible,[10,11] and, in fact, properly anticipated by a standard.[13] The idea of interval arithmetic can be extended to arithmetic of balls in Banach spaces. In particular, consider the Banach space of analytic functions of one variable in the unit disk, with real Taylor coefficients when expanded at zero, equipped with the norm

$$\|f\| = \sum_i |f_i|$$

where $f(z) = \sum_i f_i z^i$. A ball $\mathscr{B}$ in this Banach space is defined by a set of $n+1$ intervals $I_0, \ldots, I_n$ and a nonnegative number u as follows:

$$\mathscr{B}(I_0, \ldots, I_n, u) = \left\{ f \mid f_i \in I_i,\ i = 0, \ldots, n,\ \sum_{i>n} |f_i| \leqslant u \right\}$$

It is easy to see that given two balls $\mathscr{B}$ and $\mathscr{B}'$, there is a finite algorithm constructing a new ball $\mathscr{B}''$ of the same type such that

$$f + f' \in \mathscr{B}'' \qquad \text{when} \quad f \in \mathscr{B}, \quad f' \in \mathscr{B}'$$

(Take the sum of the components.) It can shown[8] that the usual arithmetic operations (such as pointwise multiplication, composition, and differentiation) are all constructed in the same way as addition. Hence, they can be programmed on a computer. In fact, every estimate of the proof outlined in this paper can be programmed, including the bound on the tangent map.

There are, however, two problems with programs of this type. The first problem is their unreadability, because current programming languages are not suitable for the kind of task needed here. The second problem is the large amount of relatively uninteresting code dealing with the operations described above.

In order to make the proofs of this paper more readable, they have been implemented on a computer as follows: First a small programming language called "Mini" has been created. "Mini" is used to program a high-level language interface to a conventional programming language (Pascal). In "Mini" the user describes the kind of Banach spaces he wants to consider. (Below we show how this is done for our particular example.) This piece of program is then handed to the computer, which generates an extension of "Pascal," called "Lang," adapted to the problem in question and allowing for straightforward notation for things like addition, multiplication, and composition of balls in function space. Furthermore, the computer also generates all the subroutines needed for the particular problem, inasmuch as the code can be inferred from the definition of the function space. This interface has been documented in detail elsewhere.[5]

We describe, informally, some of the details of this approach. First, we describe the balls in question. In our case, given n, we consider

$$\mathscr{B}(\underline{I}, u_h, u_g) = \Big\{ f \mid f(\mu, t) = \sum_{i,j} f_{ij}\mu^i t^j + g(\mu, t),$$
$$f_{ij} \in I_{ij}, \text{ for } i+j \leqslant n, \sum_{i+j>n} |f_{ij}| \leqslant u_h, \ \|g\| \leqslant u_g \Big\}$$

This is programmed by defining two types p and b for polynomials and balls, respectively. For the case of $n = 16$ one has to write the following piece of problem in the high-level language "Mini" (u is a predefined type for upper bounds, s is a predefined type for intervals):

```
type p = polynomial over s
const n=16;
var i,j:integer;
begin
 for i:=0 to n do
  for j:=0 to n-i do
   scoef:s;

end;


type b = vector over s
begin
 p:p;
 bound ug:u;
 bound uh:u;
end;
```

Also, one has to describe the product of two balls by specifying into which terms the various cross-terms are to the accumulated. In "Mini" this is programmed as follows:

```
define b * b -> b
begin
 p * p -> p;
 p * uh -> uh;
 uh * p -> uh;
 uh * uh -> uh;
 ug * ug -> ug;
 ug * uh -> uh;
 uh * ug -> uh;
 p * ug -> ug;
 ug * p -> ug;
 if it$1+it$2+imu$1+imu$2>n then
  p * p -> uh;
end;
```

Finally, the composition is prepared in "Mini" by writing the single line

```
define p o b;
```

As mentioned above, the computer produces, using the above pieces of code, a set of subroutines for the basic estimates and a "Lang" compiler (or rather, Pascal preprocessor). "Lang" allows the programmer to write the problem in more or less standard mathematical notation. The following table shows the information file, which is also generated by "Mini," and which contains everything the user needs to known in order to be able to program in "Lang":

```
INFORMATION ON LOOPS FOR STRUCTURE p
1.
We represent the component(s)
scoef
of the structure p
by array(s) [0..loop$1].
The procedure init$1 (in initloops.p) initializes these arrays.
The procedure init$$ calls all init$n.
The program calcconst.p calculates the constant
loop$1


INFORMATION ON SUBROUTINES
The call pSHOW(p1) prints p1.
The call bSHOW(b1) prints b1.
p1:=0 is implemented as p1:=pZERO
b1:=0 is implemented as b1:=bZERO
p1:=p2+p3 is implemented as p1:=pSUM(p2,p3)
p1:=p2-p3 is implemented as p1:=pDIFF(p2,p3)
```

```
b1:=b2+b3 is implemented as b1:=bSUM(b2,b3)
b1:=b2-b3 is implemented as b1:=bDIFF(b2,b3)
p1:=s2*p3 is implemented as p1:=psLMULT(s2,p3)
b1:=s2*b3 is implemented as b1:=bsLMULT(s2,b3)
p1:= -p2 is implemented as p1:=pNEG(p2)
b1:= -b2 is implemented as b1:=bNEG(b2)
u1:=|p2| is implemented as u1:=upABS(p2)
u1:=|b2| is implemented as u1:=ubABS(b2)
s1:=p2(#=x3,x4) is implemented as s1:=spVALUE(p2,x3,x4)
p2 is a polynomial evaluated at xi,
the argument(s) are of type s.
p1:=p2(#:x3,y3,x4,y4) is implemented as p1:=pDILATE(p2,x3,y3,x4,y4)
p2 is a polynomial evaluated at xi*(i'th variable)+yi.
The xi and yi are of type s.
p1:=p2*p3 is implemented as p1:=pPROD(p2,p3)
This is the truncated product of polynomials.
p1:=1/p2, the inverse of p2, is implemented as p1:=pINV(p2)
This is the truncated inverse of polynomials.
The derivative of p1 of order nimu,nit is implemented as
pDERIVE(p1,nimu,nit).
b1:=b2*b3 is implemented as b1:=bPROD(b2,b3)
b1:=b2/b3 is implemented as b1:=bQUOT(b2,b3)
b1:=1/b2, the inverse of b2, is implemented as b1:=bINV(b2)
The product with bounds is given by
the definition b * b -> b.
If type comes from polynomials then identity is defined as bONE
Composition b1:=p2(#=bimu,bit) is implemented as
b1:=bpCOMP(p2,bimu,bit)
```

As an example, we show below most of the "Lang" program computing $D\mathscr{R}$. The gain of readability and therefore checkability of the problem over the older proofs, as, e.g., in Ref. 9, is evident. A few hints may be useful. Variables start with a letter indicating their type, thus:

u... is an upper bound
s... is an interval
p... is a polynomial (with interval coefficients)
b... is a ball

The operations have their usual meaning, except that

$[u\ldots]$ is an interval whose endpoints are $u\ldots$
$[x]$, where x is an integer expression, is the interval $[x, x]$
$+-u\ldots$ is the interval $[-u\ldots, +u\ldots]$
$\langle s\ldots\rangle$ is an interval whose endpoints are the center of $s\ldots$
$|b\ldots|$ is the norm of $b\ldots$
$|s\ldots|$ is $\sup_{x\in s\ldots}|x|$

Finally, # signals substitution or evaluation in polynomials, and the meaning should be clear from the context.

3.1. The Program for Computing $D\mathscr{R}$

```
{declarations}
{n is 16}
  ...
{————————————————————————————————————————————————————}

function ubound(k,j:integer;ur:u):u;
{computes bound on j'th derivative of x**k
 for j=0,1,2
}
var i:integer;
    ures,utemp:u;
begin
if j=0 then  ubound:=ur**k
else
  begin
  if k<=j then k:=j;
   utemp:=uZERO;
   ures:=|[k]*[ur]**(k-j)|;
   if j=2 then ures:=|[ures]*[k-1]|;
   while ures.value>utemp.value do
   begin
    utemp:=ures;
    k:=k+1;
    ures:=|[ur*ures]*[k]/[k-j]|;
   end {while};
   ubound:=utemp;
  end{else};
end {ubound};
{---------------------------------------------------------------}

function uradius(u1,u2:u):u;
{takes max of arguments and checks <=1}
var umax:u;
begin
 umax:=umax2(u1,u2);
 if umax.value > 1 then
 begin
  writeln('radius too big');uSHOW(u1);uSHOW(u2);
  umax:=uONE
 end;
 uradius:=umax
end;
{---------------------------------------------------------------}

function bDILATE(barg:b;smulin,smuconst,stlin,stconst:s):b;
var bres:b;
    umax:u;
begin
 bres.p:=barg.p(#:smulin,smuconst,stlin,stconst);

  umax:=uradius(|smulin|+|smuconst|,|stlin|+|stconst|);

{higher}
  bres.uh:=uZERO;
{general}
  bres.ug:= barg.ug*ubound(0  ,0,umax)
           +barg.uh*ubound(n+1,0,umax);
bDILATE:=bres;
end;
{---------------------------------------------------------------}
```

```
function bDILATES(barg:b;smulin,stlin:s):b;
var bres:b;
    umax:u;
begin
 bres.p:=barg.p(#:smulin,sZERO,stlin,sZERO);

 umax:=uradius(|smulin|,|stlin|);

 bres.ug:=barg.ug*ubound(0   ,0,umax);
 bres.uh:=barg.uh*ubound(n+1,0,umax);
bDILATES:=bres;
end;
{————————————————————————————————————————————————————————}

function bDERIVEDILATE(barg:b;smulin,smuconst,stlin,stconst:s):b;
{1st derivative with respect to mu}
var bres:b;
    umax:u;
begin
 bres.p:=pDERIVE(barg.p,1,0)(#:smulin,smuconst,stlin,stconst);

 umax:=uradius(|smulin|+|smuconst|,|stlin|+|stconst|);

  bres.uh:=uZERO;
  bres.ug:= (barg.ug+barg.uh)*ubound(0,1,|smulin|+|smuconst|);
 bDERIVEDILATE:=bres;
end;
{————————————————————————————————————————————————————————}

function sbVALUE(barg:b;smu,st:s):s;

var umax:u;
begin
 umax:=uradius(|smu|,|st|);
 sbVALUE:=
          barg.p(#=smu,st)
    + +-(barg.ug*ubound(0   ,0,umax))
    + +-(barg.uh*ubound(n+1,0,umax));
end;

{-------------------------------------------------------------------}

function sbDERIVEVALUE(barg:b;j:integer;smu,st:s):s;
var umax:u;
begin
 umax:=uradius(|smu|,|st|);
 sbDERIVEVALUE:=
  pDERIVE(barg.p,j,0)(#=smu,st)
    + +-(barg.ug*ubound(0   ,j,umax))
    + +-(barg.uh*ubound(n+1,j,umax));
end;

{-------------------------------------------------------------------}

function bCOMP(barg,bmu,bt:b):b;
var bres:b;
     umax:u;
begin
 bres:=barg.p(#=bmu,bt);
 umax:=uradius(|bmu|,|bt|);
 bres.ug:=bres.ug
              +barg.ug*ubound(0   ,0,umax)
              +barg.uh*ubound(n+1,0,umax);

 bCOMP:=bres;
end;

{-------------------------------------------------------------------}

function bDERIVECOMP(barg,bmu,bt:b):b;
{differentiate w.r.t. t}
```

```
var bres:b;
    umax:u;
begin
 bres:=pDERIVE(barg.p,0,1)(#=bmu,bt);
 umax:=uradius(|bmu|,|bt|);

 bres.ug:=bres.ug
           +barg.ug*ubound(0   ,1,umax)
           +barg.uh*ubound(n+1,1,umax);

 bDERIVECOMP:=bres;
end;
{——————————————————————————————————————————————————————————————}

function bFtoH(bF:b):b;
var btemp:b;
begin
 btemp:=bZERO;
 btemp.p.scoef[0,0]:=salpha;
 btemp.p.scoef[0,1]:=srho;
 bFtoH:=bONE+btemp*bF;
end;
{——————————————————————————————————————————————————————————————}

function bHtoF(bH:b):b;
var pnew,ptemp:p;
    bres,btemp,bt:b;
    i,j,k:integer;
begin
 ptemp:=bH.p-pONE;
 for i:=0 to n do
  for j:=0 to n-i do
   ptemp.scoef[i,j]:=
     <ptemp.scoef[i,j]>;
 pnew:=pZERO;
```

```
{compute (rho+alpha/t)**(-1)*(bH.p-1)}
 for i:=0 to n do
  for j:=0 to n-i do
   for k:=0 to n-i-j do
     pnew.scoef[i,j]:=
      pnew.scoef[i,j]+
       ptemp.scoef[i,j+k]*(st0**k);


{divide by rho times t}
ptemp:=pZERO;
for i:=0 to n  do
 for j:=1 to n-i do
  ptemp.scoef[i,j-1]:=
   <pnew.scoef[i,j]/srho>;


 btemp:=bZERO;
 btemp.p:=ptemp;
 bt:=bZERO;
 bt.p.scoef[0,0]:=salpha;
 bt.p.scoef[0,1]:=srho;
 bH:=bH-btemp*bt-bONE;
 bres.p:=ptemp;
 bres.ug:=|[|bH|]/(srho-[|salpha|])|;
 bres.uh:=uZERO;

 bHtoF:=bres;
end;
{-------------------------------------------------------------}

procedure newton;
var sepsilon,snew,sbeta:s;
    i:integer;
begin
{find approximate root}
```

```
{known guess}
 smu0:=sZERO;
 for i:=1 to 20 do
 begin
  sepsilon:=bH4.p(#=smu0,st1)-slambdabar;
  stau:=pDERIVE(bH4.p,1,0)(#=smu0,st1);
  smu0:=smu0-sepsilon/stau;
  smu0:=<smu0>;
 end;


{compute values at central guess}
 sepsilon:=sbVALUE(bH4,smu0,st1)-slambdabar;


{increase interval to contain 0}
 sepsilon.lower:=lmin2(sepsilon.lower,lZERO);
 sepsilon.upper:=umax2(sepsilon.upper,uZERO);



 snew:= +- unCONST(7.0E-02);
 repeat
  sbeta:=snew;
  stau:=sbDERIVEVALUE(bH4,1,smu0+sbeta,st1);
  snew:=-sepsilon/stau;
  if not EQUALs(sINTER(snew,sbeta),snew) then
  begin
   writeln('no contraction for ZERO');
   sSHOW(snew);
  end;
 until EQUALs(snew ,sbeta);

 smu0:=smu0+sbeta;

end;
{------------------------------------------------------------------}
```

```
procedure tangent_map;
begin

bdlambda:=bDILATES(bdF,ssigmabar,sZERO);
bdH :=(srho*bt+salpha*bONE)*bdF;
bdF1:=bCOMP(bdF,ssigmabar*bmu,btarg1)+
       [2]*bdlambda*blambda
         *bDERIVECOMP(bF ,ssigmabar*bmu,btarg1)*btrho;
bdF2:=blambda*
       (bdF1*blambda+[2]*bdlambda*bF1);

bdF3:=[2]*bdF2*bFtoH(bF2);


bdH3:=(srho*bt+salpha*bONE)*bdF3;


bdH4:=blambdainv*
       (-bdlambda*bH4
        +stt*bdH3*bDERIVECOMP(bH,ssigmabar*bmu,btarg2)
        +bCOMP(bdH,ssigmabar*bmu,btarg2)
       );
sdmu0:=-sbVALUE(bdH4,smu0,st1)/stau;
sdtau:=sbDERIVEVALUE(bdH4,1,smu0,st1)+
        sbDERIVEVALUE(bH4 ,2,smu0,st1)*sdmu0;


bdH5:= bDILATE(bdH4,sepsilonbar/stau,smu0,sONE,sZERO)
        +((-sdtau*sepsilonbar/stau**2)*bmu
           +sdmu0*bONE
         )*bDERIVEDILATE(bH4,sepsilonbar/stau,smu0,sONE,sZERO);



bdRF:=bHtoF(bdH5+bONE
             +(srho*bt+salpha*bONE)
                *((slambdabar-sONE)*bONE+sepsilonbar*bmu)
            )
       -(slambdabar-sONE)*bONE-sepsilonbar*bmu;
```

```
writeln('norm=');uSHOW(|bdRF|);
uresult:=umax2(uresult,|bdRF|);
end;

{————————————————————————————————————————————————————————}

begin {main}

{the function mu, and t}
bmu:=bZERO;
bmu.p.scoef[1,0]:=sONE;
bt:=bZERO;
bt.p.scoef[0,1]:=sONE;

{coordinates}
salpha:=[1];
srho:=[2];
ssigmabar:=sCONST(0.23);
sepsilonbar:=sCONST(0.0005);
slambdabar:=sCONST(-1.0/2.5029078750957);

{the function t+1/rho}
btrho:=bZERO;
btrho.p.scoef[0,0]:=sONE/srho;
btrho.p.scoef[0,1]:=sONE;

{the points 0,1, and the coefficient of t}
st0:=-salpha/srho;
st1:=(sONE-salpha)/srho;
stt:=sONE/srho;

{read in data and make bF}
{%include 'coefs.16.ins';}
    ...
```

```
{corresponding H}
bH:=bFtoH(bF);

{lambda(mu) = f(scale*mu,1)}
blambda:=bDILATES(bF,ssigmabar,sZERO)+bONE;
blambdainv:=1/blambda;
blambda2:=blambda*blambda;

btarg1:=blambda2*(bt-st0*bONE)+st0*bONE;
bF1:=bCOMP(bF,ssigmabar*bmu,btarg1);
bF2:=blambda2*bF1;
bF3:=bF2*((srho*bt+salpha*bONE)*bF2+[2]*bONE);


bH3:=bFtoH(bF3);
btarg2:=stt*bH3+st0*bONE;
bH4:=bCOMP(bH,ssigmabar*bmu,btarg2)*blambdainv;
newton;


bH5:=bDILATE(bH4,sepsilonbar/stau,smu0,sONE,sZERO);
bRF:=bHtoF(bH5);


writeln('R-difference=');uSHOW(|bRF-bF|);
writeln('bRF');bSHOW(bRF);
uresult:=uZERO;


 writeln('higher');
 bdF:=bZERO;
 bdF.uh:=uONE;
 tangent_map;


for i:=0 to n do
 for j:=0 to n-i do
 if (i+j>=2) or ((i=0) and (j=1)) then
 begin
  writeln(i,j);
  bdF:=bZERO;
  tangent_map;
 end;
 writeln('NORM OF DR');uSHOW(uresult);

end {main};
```

ACKNOWLEDGMENTS

We have profited from the warm hospitality of the Zentrum für Theoretische Studien (ETH Zürich) and of the Center for Mathematical Sciences Research (New Brunswick), which made this collaboration possible. We also would like to thank A. Malaspinas and S. Oliffson Kamphorst for many helpful discussions. This work was supported in part by the Fonds National Suisse (to J.-P.E. and P.W.) and National Science Foundation grants DMR-81-14726 (to J.-P.E. and P.W.) and DMS-85-18622 (to P.W.).

REFERENCES

1. M. J. Feigenbaum, Quantitative universality for a class of nonlinear transformations, *J. Stat. Phys.* **19**:25–52 (1978); **21**:669–706 (1978).
2. P. Collet, J.-P. Eckmann, and O. E. Lanford III, Universal properties of maps on the interval, *Commun. Math. Phys.* **76**:211–254 (1980).
3. E. B. Vul and K. M. Khanin, The unstable separatrix of Feigenbaum's fixed-point, *Russ. Math. Surveys* **37**(5):200–201 (1982).
4. E. B. Vul, Ya. G. Sinai, and K. M. Khanin, Feigenbaum universality and the thermodynamic formalism, *Russ. Math. Surveys* **39**(3):1–40 (1984).
5. J.-P. Eckmann, A. Malaspinas, and S. Oliffson Kamphorst, to be published.
6. O. E. Lanford III, A computer-assisted proof of the Feigenbaum conjectures, *Bull. AMS* N. S. **6**:127 (1984).
7. H. Koch and P. Wittwer, A non-Gaussian renormalization group fixed point for hierarchical scalar lattice field theories, *Commun. Math. Phys.*, to appear.
8. J.-P. Eckmann, H. Koch, and P. Wittwer, A computer-assisted proof of universality for area-preserving maps, *Mem. AMS* **47**:289 (1984).
9. J.-P. Eckmann and P. Wittwer, Computer methods and Borel summability applied to Feigenbaum's equation, *Lecture Notes in Physics* (Springer-Verlag, Berlin, 1985).
10. R. E. Moore, *Interval Analysis* (Prentice-Hall, 1966).
11. R. E. Moore, *Methods and Applications of Interval Analysis* (SIAM, Philadelphia, 1979).
12. R. de la Llave and O. E. Lanford III, to be published.
13. D. Stevenson, IEEE Computer Society. A proposed standard for binary floating-point arithmetic, Draft 8.0 of IEEE Task P754, Computer, 51-62 (March 1981).