Journal of Statistical Physics, Vol. 89, Nos. 5/6, 1997

Calculating Topological Entropy

Stewart L. Baldwin[1] and Edward E. Slaminka[1]

Received March 5, 1997

The attempt to find effective algorithms for calculating the topological entropy of piecewise monotone maps of the interval having more than three monotone pieces has proved to be a difficult problem. The algorithm introduced here is motivated by the fact that if f: $[0, 1] \to [0, 1]$ is a piecewise monotone map of the unit interval into itself, then $h(f) = \lim_{n \to \infty} (1/n) \log \operatorname{Var}(f^n)$, where $h(f)$ is the topological entropy of f, and $\operatorname{Var}(f^n)$ is the total variation of f^n. We show that it is not feasible to use this formula directly to calculate numerically the topological entropy of a piecewise monotone function, because of the slow convergence. However, a close examination of the reasons for this failure leads ultimately to the modified algorithm which is presented in this paper. We prove that this algorithm is equivalent to the standard power method for finding eigenvalues of matrices (with shift of origin) in those cases for which the function is Markov, and present encouraging experimental evidence for the usefulness of the algorithm in general by applying it to several one-parameter families of test functions.

KEY WORDS: Topological entropy; algorithm; power method; Markov map.

1. INTRODUCTION

Topological entropy has been shown to be effective at predicting complex behavior in dynamical systems. Since it was first defined in 1965 by Adler, Konheim and McAndrew,[1] much progress has been made in the theoretical implications of this concept (cf. [5] and [11] for example). However, the numerical calculation of the entropy for specific examples has proved to be a difficult problem. Many researchers have proposed numerical algorithms for approximating the entropy, most of which have been for maps of the interval. The seminal paper of Milnor and Thurston[10] developed the concept of kneading sequences for understanding iterated maps of the

[1] Department of Mathematics, Auburn University, Auburn, Alabama 36849-5310.

interval. In 1983, Collet, Crutchfield and Eckmann[7] proved that the topological entropy of a unimodal map depended only on the kneading sequence of its critical point, and used this to present an algorithm for calculating the entropy of such maps. Block, Keesling, Li and Peterson,[4] in 1989, presented a fast and efficient algorithm for computing the entropy of unimodal maps of the interval. In 1992, Block and Keesling[3] extended that result to maps with three monotone pieces. Góra and Boyarsky,[8] in 1991, developed an algorithm for not necessarily continuous piecewise monotone maps of the interval. However, their algorithm is not very efficient nor even accurate (cf. Figs. 1 and 2 of [8]). In 1993, Newhouse and Pignataro[13] devised two algorithms to compute the entropy. Though the rates of convergence are extremely slow and memory usage is prohibitively high, it should be noted that these algorithms are intended to compute the entropy for not only one-dimensional dynamical systems, but for arbitrarily high dimensions. Balmforth, Spiegel and Tresser,[2] in 1994, based their algorithm on the class of Markov maps (defined below), for which the calculation of entropy is just an eigenvalue calculation of an associated matrix, and used this idea to motivate a more general algorithm for all piecewise monotone maps of the interval. The original motivation for the algorithms described in this paper comes from the following result of Misiurewicz and Szlenk.[12]

Theorem 1. If f is a piecewise monotone continuous function on the unit interval, then $h(f) = \lim_{n\to\infty}(1/n) \log \operatorname{Var}(f^n)$, where $h(f)$ denotes the topological entropy of f, and Var denotes the total variation.

Given that $\operatorname{Var}(f^n)$ is a value which can be calculated in many cases, this formula suggests a possible method for calculating the topological entropy of a piecewise monotone map of the interval. Whether or not this method is practical depends on two factors. First, it must be possible to calculate $\operatorname{Var}(f^n)$ in a way that does not grow too quickly as a function of n. The most obvious way of calculating $\operatorname{Var}(f^n)$ is to find all of the turning points of f^n and then calculate the variation on each of these intervals. However, the number of turning points grows exponentially with n whenever $h(f) > 0$, making this a very inefficient way of calculating $\operatorname{Var}(f^n)$. (This is also the problem with the Góra–Boyarsky algorithm,[8] which requires the calculation of all such turning points.) However, we shall show that there is another method for calculating $\operatorname{Var}(f^n)$ such that the amount of time required grows only as a polynomial in n, making the calculation of $\operatorname{Var}(f^n)$ feasible for n as large as 200 in the examples considered as test functions in this and other papers.

The other reqirement for practicality is that the estimate $(1/n) \log \operatorname{Var}(f^n)$ should converge rapidly enough to the entropy to be a reasonable estimate for values of n for which $\mathrm{Var}(f^n)$ can be reasonably calculated. Unfortunately, as will be shown below, there are trivial examples of piecewise monotone functions for which the convergence is as bad as $O(1/n)$, making the direct use of this formula impractical.

However, even though the attempt to use the above formula directly is not feasible, there is a modification which shows more promise. It was observed that, in many cases, $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$ converges very rapidly to $h(f)$. It is relatively easy to prove that $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$, if it converges at all, must converge to $h(f)$, but there are examples where this sequence does not converge, so it cannot be used directly as a method for calculating $h(f)$. However, close examination of a class of maps called the Markov maps gives a clear picture of why the formula $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$ sometimes fails to converge, and suggests a modification which always converges to the entropy for Markov maps (usually with error $O(c^n)$ for some $c < 1$), and gives excellent experimental results on those functions on which it has been tested. As will be shown below, this method calculates a weighted average $b_n = \sum_{i=0}^{n} \binom{n}{i} \mathrm{Var}(f^i)$, and then uses $\log(b_{n+1}/b_n - 1)$ as the nth-stage approximation to $h(f)$, and in the Markov case can be shown to be equivalent to the Power Method for finding eigenvalues of a matrix, with shift of origin.

The remainder of Section 1 will introduce the basic definitions. In Section 2, we establish a simple polynomial time algorithm for calculating $\mathrm{Var}(f^n)$ for any piecewise monotone function f. Section 3 will cover the Markov case in detail, using that case to motivate the eventual algorithm which will then be applied to the more general case. Section 4 presents the numerical results when this algorithm is applied to a number of examples, and Section 5 presents our conclusions.

Definition. Let I be a closed interval of real numbers. A continuous function $f: I \to I$ is called *piecewise monotone* iff there is a finite collection of subintervals of I whose union is I, with f monotone on each of these subintervals. (Here, monotone means that inverse images of points are connected, so we are not necessarily assuming strict monotonicity.) A *turning interval* is any interval consisting entirely of local extrema which is maximal with respect to that property. Clearly, a continuous function on I is piecewise monotone iff it has finitely many turning intervals. If a turning interval is a singleton, then the resulting point is called a *turning point*. For the remainder of this paper, we shall assume that every turning interval is a singleton, and refer to turning points instead. However, this is merely a notational convenience, and it is easily seen that the results of this paper can be modified in a trivial way to include piecewise monotone maps with nontrivial turning intervals.

A piecewise monotone map $f: I \to I$ is called *Markov* iff there is a finite set $P \subseteq I$ which contains all turning points and both endpoints, such that $f(P) \subseteq P$. In this case, a *P-basic interval* is defined to be the closure of any component of $I \setminus P$. If we let G be the set of P-basic intervals, then we define a directed graph on G by defining the binary relation $\to$ by $I \to J$ iff $J \subseteq f(I)$. If $G = \{I_1, ..., I_n\}$, then the *incidence matrix* of G can be defined as $A = (a_{ij})$ where $a_{ij} = 1$ if $I_i \to I_j$ and $a_{ij} = 0$ otherwise. Then it is well known that $h(f)$ is equal to the log of largest positive eigenvalue of the matrix A (cf. [5]).

2. CALCULATING THE TOTAL VARIATION

In this section we show that there is a simple polynomial (in n) time algorithm for calulating $\mathrm{Var}(f^n)$ in the case where f is piecewise monotone.

If $h(f) > 0$, then the number of turning points of f^n grows exponentially with respect to n. In fact the exponential rate of growth is very roughly of the form a^n, where $h(f) = \log a$. Thus, any method which requires knowing all of the turning points of f^n is going to require an unreasonably large number of calculations. Fortunately, we do not need to know this information. The key trick is motivated by the fact that, while the set $T_n = \{x : x \text{ is a turning point of } f^n\}$ grows exponentially, the set $f^n(T_n)$ only grows linearly, that is, $f^n(T_n) \subseteq \bigcup_{i=1}^{n} f^i(T_1)$, and that to calulate $\mathrm{Var}(f^n)$, we only need to calculate the number of times each component of $I \setminus f^n(T_n)$ is covered by f^n.

Definition. If P is a finite subset of I, and we write $P = \{p_0, p_1, ..., p_n\}$, then we also mean that $p_0 < p_1 < \cdots < p_n$, and that p_0 and p_n are the endpoints of I, unless otherwise stated. If P is a finite set, we use $|P|$ to denote the number of elements of P. We say that a triple (f, P, Q) is *compatible* if and only if $f: I \to I$ is piecewise monotone, P and Q are finite subsets of I such that $f(P) \subseteq Q$, and $\{f(x): x \text{ is a turning point of } f\} \subseteq Q$. If (f, P, Q) is compatible, where $P = \{p_0, p_1, ..., p_n\}$ and $Q = \{q_0, q_1, ..., q_m\}$, then $\mathbf{A}_f(Q, P)$ is defined to be the m by n matrix $\mathbf{A} = (a_{ij})$, where for each pair (i, j), a_{ij} is the maximum possible number of pairwise disjoint intervals $J \subseteq (p_{j-1}, p_j)$ such that $f(J) = (q_{i-1}, q_i)$, i.e., a_{ij} is the number of components of $f^{-1}(q_{i-1}, q_i)$ which are contained in (p_{j-1}, p_j). We also define $\mathbf{c}_f(Q)$ to be the vector $(c_1, c_2, ..., c_m)$, where for each i, c_i is the maximum possible number of pairwise disjoint intervals $J \subseteq I$ such that $f(J) = (q_{i-1}, q_i)$, i.e., c_i is the number of components of $f^{-1}(q_{i-1}, q_i)$. Given P as above, $\mathbf{d}_P$ is defined to be the vector $(p_1 - p_0, p_2 - p_1, ..., p_n - p_{n-1})$. We define $\mathbf{1}_n$ to be the n-dimensional vector, all of whose components are 1, with the subscript suppressed if it is obvious from context.

Proposition 1. If (f, P, Q) is a compatible triple, with $|P| = n$ and $|Q| = m$, then $\mathbf{c}_f(Q) = \mathbf{A}_f(Q, P)\, \mathbf{1}_n$.

Proposition 2. If (f, P, Q) is a compatible triple, then $\operatorname{Var}(f) = \mathbf{d}_Q \mathbf{c}_f(Q)$, where $\operatorname{Var}(f)$ is the total variation of f.

Lemma 1. If (f, P, Q) and (g, Q, R) are both compatible triples, then $(g \circ f, P, R)$ is also a compatible triple, and $\mathbf{A}_{g \circ f}(R, P) = \mathbf{A}_g(R, Q)\, \mathbf{A}_f(Q, P)$.

Proof. Suppose $|P| = p$, $|Q| = q$, and $|R| = r$. Let (a_{ij}) represent the entries of the matrix $\mathbf{A}_g(R, Q)$, and let (a'_{ij}) represent the entries of $\mathbf{A}_f(Q, P)$. If x is a turning point of $g \circ f$, then either x is a turning point of f or $f(x)$ is a turning point of g, and in either case it is easy to see from the definition of a compatible triple that $g(f(x)) \in R$. Thus $(g \circ f, R, P)$ is a compatible triple. Fix i and j, with $1 \leqslant i \leqslant r$ and $1 \leqslant j \leqslant p$. Then for each k, $1 \leqslant k \leqslant q$, and for each $x \in (r_{i-1}, r_i)$, $g^{-1}(x)$ has exactly a_{ik} components lying in the interval (q_{k-1}, q_k), and for each $y \in (q_{k-1}, q_k)$, $f^{-1}(y)$ has exactly a'_{kj} components lying in the interval (p_{j-1}, p_j). Thus, for each such x, there are exactly $a_{ik} a'_{kj}$ components C of $(G \circ f)^{-1}(x)$ such that $f(C) \subseteq (q_{k-1}, q_k)$, from which it easily follows that the ij-entry of $\mathbf{A}_{g \circ f}(R, P)$ is $\sum_{k=1}^{q} a_{ik} a'_{kj}$.

Corollary 1. Suppose (f, P_0, P_1) is a compatible triple, and P_{n+1} is inductively defined by $P_{n+1} = P_n \cup f(P_n)$ for $n > 1$. Then for any $m < n$, (f^{n-m}, P_m, P_n) is a compatible triple, and $\mathbf{A}_{f^{n-m}}(P_n, P_m) = \mathbf{A}_f(P_n, P_{n-1}) \cdots \mathbf{A}_f(P_{m+1}, P_m)$.

Corollary 2. Given f, P_n as in the previous Corollary, $\operatorname{Var}(f^n) = \mathbf{d}_{P_n} \mathbf{A}_f(P_n, P_{n-1}) \cdots \mathbf{A}_f(P_2, P_1)\, \mathbf{A}_f(P_1, P_0)\, \mathbf{1}$.

The calculation of $\operatorname{Var}(f^n)$ can be made more efficient if one observes the following. The matrix $\mathbf{A}_f(P_{n+1}, P_n)$ is constructed by noting that the row partition of $\mathbf{A}_f(P_{n+1}, P_n)$ is the column partition of $\mathbf{A}_f(P_n, P_{n-1})$ and the column partition of $\mathbf{A}_f(P_{n+1}, P_n)$ is constructed from the set $P_n \cup f(P_n)$. We can also rewrite $\operatorname{Var}(f^n)$ as $\mathbf{d}_{P_n} \mathbf{c}_{f^n}(P_n)$. The term $\mathbf{c}_{f^n}(P_n)$ can be calculated so as to be a k by 1 matrix, for some k. Thus by Corollary 2, $\operatorname{Var}(f^{n+1}) = \mathbf{d}_{P_{n+1}} \mathbf{A}_f(P_{n+1}, P_n)\, \mathbf{c}_{f^n}(P_n)$. At each iteration we perform one matrix multiplication by a vector followed by a dot product. In effect, we use the results of the computation for the variation of f^n to compute the variation of f^{n+1}.

3. REFINEMENTS OF THE METHOD

Given that there is a simple polynomial-time algorithm for calculating $\mathrm{Var}(f^n)$, as described in the previous section, one can then ask if there is a reasonable way to convert it into an algorithm for approximating $h(f)$ in a reasonable amount of time. A simple example suffices to show that the usual formula $h(f) = \lim_{n\to\infty}(1/n)\log(\mathrm{Var}(f^n))$ converges too slowly to be useful, even in some very simple cases.

For example let $I = [0, 2]$, and define $f\colon I \to I$ by $f(x) = 2x$ when $x \in [0, 1]$, and $f(x) = -2x + 4$ when $x \in [1, 2]$, i.e., the usual "tent map" based on the interval $[0, 2]$. Then it is easy to see that $h(f) = \log 2$ and $\mathrm{Var}(f^n) = 2^{n+1}$. Therefore, the error E_n in using $(1/n)\log(\mathrm{Var}(f^n))$ to calculate $h(f)$ is $E_n = (1/n)\log(\mathrm{Var}(f^n)) - h(f) = (1/n)\log(2^{n+1}) - \log 2 = (\log 2/n)$, which clearly converges to zero too slowly to be of any practical value in a calculation which requires several significant digits.

Scaling to the unit interval would not help this problem, for it is easy to construct functions on $[0, 1]$ having a small invariant interval with large entropy, and equal to the identity outside that interval. Such a function would have the same problems as above. With a little more work, one can find such functions for which the relevant invariant set (which need not be an interval) would not be readily apparent, so that the scaling factor needed to avoid the above problem would be unknown.

Note that in the special case that $f(P_0) = P_0$, we have that $P_n = P_0$ for all n, and the matrices $\mathbf{A}_f(P_{n+1}, P_n)$ are all the same. Call this matrix $\mathbf{A}$, and let $P = P_0$. Then in this case we get $\mathrm{Var}(f^n) = \mathbf{d}_P\mathbf{A}^n\mathbf{1}$ for all n, and the algorithm becomes virtually identical to the *Power Method*, one of the typical methods for calculating the dominant eigenvalue of the matrix $\mathbf{A}$. If each point of P_1 is eventually periodic, then we have that $P_n = P_N$ for all n greater than some fixed N, and we could assume that $P_0 = P_N$, since the result of the algorithm would be the same from the Nth step on. Thus, it is useful to see what happens to the convergence of these extimates in that case.

If the usual power method were used to find the dominant eigenvalue of the matrix $\mathbf{A}$, then a typical procedure might be to start with the vector $\mathbf{v}_0 = \mathbf{1}$, and recursively define $v_n = |\mathbf{v}_n|$ and $\mathbf{v}_{n+1} = (1/v_n)\mathbf{A}\mathbf{v}_n$. Thus, in the discussion which is given below, it should be kept in mind that, in the Markov case, the method being used here is equivalent to using the standard power method for calculating eigenvalues, and that further details on the rate of convergence can be obtained from standard references on the subject (cf. [6] or [14], for example). In many cases (the exceptions to which are discussed below) it is expected that the numbers v_n will converge to the dominant eigenvalue λ. Since $h(f) = \log\lambda$ in this case, it is easily seen that this would be similar to using $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$ rather than $(1/n)\log(\mathrm{Var}(f^n))$ as an estimate for $h(f)$. Now, the sequence $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$ need not converge (as examples below will show), but if it does, it must converge to $h(f)$, an immediate consequence of the following well known fact.

Proposition 3. Let $\langle c_n \rangle$ be a sequence of positive real numbers. If the sequence $\langle c_{n+1}/c_n \rangle$ converges, then so does the sequence $\langle (c_n)^{1/n} \rangle$, and to the same limit.

Outline of Proof. If $L = \lim_{n\to\infty}(c_{n+1}/c_n)$, and $K < L < M$, then there are positive constants a and b such that $aK^n < c_n < bM^n$ for all n, and therefore $a^{1/n}K < (c_n)^{1/n} < b^{1/n}M$ for all n. ▮

The practical advantage of using $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$ to estimate $h(f)$ is that when it converges, it tends to converge to its limit much more rapidly than $(1/n)\log(\mathrm{Var}(f^n))$.

Let us first consider the case where the matrix $\mathbf{A}$ is diagonalizable. If we let $\lambda_1, \lambda_2, \ldots, \lambda_p$ be the eigenvalues of $\mathbf{A}$, with λ_1 the dominant positive eigenvalue, then we get

$$(*)\quad \mathrm{Var}(f^n) = c_1\lambda_1^n + c_2\lambda_2^n + \cdots + c_p\lambda_p^n,$$

where the c_i's are constants, and $c_1 \neq 0$. Thus, if C is any positive real number which is at least as large as the absolute values of all of the eigenvalues $\lambda_2, \lambda_3, \ldots, \lambda_p$, then it easy to see that there is a positive constant a such that $|\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n) - \lambda_1| < a(C/\lambda_1)^n$. Thus, if all of the other eigenvalues have absolute value strictly less than λ_1, then $C < \lambda_1$, and it is easy to see that the estimate $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$ converges exponentially to $h(f)$.

Now, suppose that $\lambda_2 = -\lambda_1$, with all other eigenvalues having smaller absolute value. (A simple example where this occurs is the function f on the interval $[0, 3]$ defined by $f(0) = f(2) = 2$, $f(1) = 3$, and $f(3) = 0$, with f piecewise linear in between.) Then the dominant terms of $\mathrm{Var}(f^n)$ will be of the form $c_1\lambda_1^n + c_2\lambda_2^n$, with $|c_2| < |c_1|$. In this case, $\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n)$ does not converge. However, there is still a well-known trick (called "shift-of-origin") for dealing with this situation. First, we note that the matix $\mathbf{A}$ has no negative entries, and therefore $\rho(\mathbf{A}) = \max\{|\lambda|: \lambda \text{ is an eigenvalue of } \mathbf{A}\}$ is itself an eigenvalue of $\mathbf{A}$ (cf. [9], Theorem 8.3.1, for example). As is well known, the eigenvalues of the matrix $\mathbf{A} + \mathbf{I}$ are $\{\lambda + 1: \lambda \text{ is an eigenvalue of } \mathbf{A}\}$. Now, if λ_1 is a positive real number, and all other eigenvalues of $\mathbf{A}$ have absolute value less than or equal to λ_1 (which is always the case in the matrices considered here), then the eigenvalues of $\mathbf{A} + \mathbf{I}$ other than $\lambda_1 + 1$ will all have absolute value strictly less than $\lambda_1 + 1$. Thus, instead of using the incidence matrix $\mathbf{A}$ of the Markov Graph of f, we use the matrix $\mathbf{A} + \mathbf{I}$ instead. Its dominant eigenvalue will be approached exponentially by the algorithm, after which we can simply subtract 1 to get the original desired eigenvalue. In fact, the standard shift-of-origin trick for finding eigenvalues of a matrix uses $\mathbf{A}+t\mathbf{I}$ for some constant t, and it is possible that an appropriately clever choice of t will increase the rate of convergence. For the purposes of the algorithm given here, we have used $t=1$.

Of course, for the Markov case, the entire discussion above is just part of the standard bag of tricks for calculating eigenvalues. What we now want to ask is: What about the case where the piecewise monotone function is not Markov? In that case, there is a natural way of taking the matrix $\mathbf{A}+\mathbf{I}$ and generalizing it to the matrices $\mathbf{A}_f(Q, P)$. The matrix $\mathbf{A}_\imath(Q, P)$ is well defined for any partitions P and Q such that $f(P) \subseteq Q$, where we let $\imath$ be the identity function on the interval. Thus, even though the proof of convergence which was discussed above does not go through in this case, there is still an obvious way to adjust the algorithm to take advantage of this idea. Thus, the suggested modification of the algorithm to deal with this case is as follows:

1. For each compatible triple (f, P, Q), let $\mathbf{B}_f(Q, P)=\mathbf{A}_f(Q, P)+\mathbf{A}_\imath(Q, P)$.
2. Do the above algorithm, using $\mathbf{B}_f(P_{n+1}, P_n)$ instead of $\mathbf{A}_f(P_{n+1}, P_n)$, producing a number $V_n=\mathbf{d}_{P_n}\mathbf{B}_f(P_n, P_{n-1})\cdots\mathbf{B}_f(P_2, P_1)\,\mathbf{B}_f(P_1, P_0)\,\mathbf{1}$.
3. Let $\log(V_{n+1}/V_n-1)$ be the corresponding nth-stage approximation of the entropy $h(f)$.

Definition. We let the *Basic Algorithm* refer to the procedure which uses $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$ as the nth-stage approximation for $h(f)$, and the *Revised Algorithm* refers to the use of $\log(V_{n+1}/V_n-1)$ as the nth-stage approximation.

As already discussed above, there are Markov functions for which the Basic Algorithm fails to converge to $h(f)$, but the Revised Algorithm does converge to $h(f)$. Although we have been unable to prove that the Revised Algorithm converges in all cases, we can show that it converges whenever the Basic Algorithm does.

Lemma 2. Let V_n be as above in the description of the Revised Algorithm. Then $V_n=\sum_{i=0}^{n}\binom{n}{i}\mathrm{Var}(f^i)$.

Proof. A simple proof by induction, very similar to the proof of the Binomial Theorem.

Lemma 3. Suppose that $\langle a_n\rangle$ is a sequence of positive real numbers such that $\lim_{n\to\infty} a_n=L>0$, and $\lim_{n\to\infty}(a_{n+1}/a_n)^n=1$. For each positive integer n, let $b_n = \sum_{i=0}^{n} \binom{n}{i} a_i^i$. Then $\lim_{n\to\infty} (b_{n+1}/b_n)$ exists and is equal to $L+1$.

Proof. Let $\varepsilon > 0$ be arbitrary. We use the convention that $\binom{n}{n+1} = \binom{n}{-1} = 0$, and recall the identity $\binom{n+1}{i} = \binom{n}{i} + \binom{n}{i-1}$. Since $a_{n+1}^{n+1}/a_n^n = (a_{n+1}/a_n)^n \, a_{n+1}$, the hypotheses of the lemma imply that there is a positive integer N such that $a_{n+1}^{n+1}/a_n^n - L < \varepsilon/2$ whenever $n \geqslant N$. Fix such an N. For each n, let $p(n) = \sum_{i=0}^{N} \binom{n}{i}(a_{n+1}^{n+1}/a_n^n - L)$, and observe that $p(n)$ is a polynomial in n with positive coefficients. Since $L > 0$ and all a_n's are positive, there is an $\eta > 0$ such that $a_n \geqslant \eta$ for all n, and thus $b_n \geqslant \sum_{i=0}^{n} \binom{n}{i} \eta^i = (1+\eta)^n$ for all n. Let $M > N$ be a positive integer such that $p(n)/(1+\eta)^n < \varepsilon/2$ whenever $n \geqslant M$. Then for any $n \geqslant M$, we have

$$\begin{aligned}
\frac{b_{n+1}}{b_n} - (L+1) &= \frac{1}{b_n}(b_{n+1} - (L+1)\, b_n) \\
&= \frac{1}{b_n}\left(\sum_{i=0}^{n+1} \binom{n+1}{i} a_i^i - (L+1) \sum_{i=0}^{n} \binom{n}{i} a_i^i\right) \\
&= \frac{1}{b_n}\left(\sum_{i=0}^{n+1} \binom{n}{i} a_i^i + \sum_{i=0}^{n+1} \binom{n}{i-1} a_i^i \right.\\
&\qquad \left. - L \sum_{i=0}^{n} \binom{n}{i} a_i^i - \sum_{i=0}^{n} \binom{n}{i} a_i^i\right) \\
&= \frac{1}{b_n}\left(\sum_{i=-1}^{n} \binom{n}{i} a_{i+1}^{i+1} - L \sum_{i=0}^{n} \binom{n}{i} a_i^i\right) \\
&= \frac{1}{b_n} \sum_{i=0}^{n} \binom{n}{i}\left(\frac{a_{i+1}^{i+1}}{a_i^i} - L\right) a_i^i \\
&= \frac{1}{b_n}\left(\sum_{i=0}^{N} \binom{n}{i}\left(\frac{a_{i+1}^{i+1}}{a_i^i} - L\right) a_i^i \right.\\
&\qquad \left. + \sum_{i=N+1}^{n} \binom{n}{i}\left(\frac{a_{i+1}^{i+1}}{a_i^i} - L\right) a_i^i\right) \\
&\leqslant \frac{1}{b_n}\left(p(n) + \sum_{i=N+1}^{n} \binom{n}{i} \frac{\varepsilon}{2} a_i^i\right) \\
&< \frac{p(n)}{(1+\eta)^n} + \frac{\varepsilon}{2} \leqslant \varepsilon
\end{aligned}$$

Corollary 3. Let f be a piecewise monotone map of the interval I such that for some nontrivial subinterval $J \subset I$, $J = f(J)$. If the Basic Algorithm converges, then so does the Revised Algorithm.

Proof. Let $a_n = (\mathrm{Var}(f^n))^{1/n}$ for each n. Then, since $\mathrm{Var}(f^n)$ is at least the length of J, $\lim_{n\to\infty} a_n = L \geqslant 1$, where $L = \exp(h(f))$. Furthermore, since the Basic Algorithm converges, we also know that $\lim_{n\to\infty}(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n)) = L$, and thus $\lim_{n\to\infty}(a_{n+1}/a_n)^n = \lim_{n\to\infty}(1/a_{n+1})(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n)) = (1/L)\,L = 1$. Thus, all of the hypotheses of Lemma 3 hold, and $\lim_{n\to\infty}(b_{n+1}/b_n) = L+1$, and we are done, since $\log(b_{n+1}/b_n - 1)$ is exactly the nth term estimate for the entropy in the revised algorithm. ∎

We finish this section with a discussion of the case in which the function is Markov, and the corresponding incidence matrix is not diagonalizable. Consider the following example. Let $f\colon [0,2] \to [0,2]$ be the function defined by $f(x) = 2x$ for $x \in [0,1]$, and $f(x) = 3 - x$ for $x \in [1,2]$. Let $P_0 = \{0, 1, 2\}$. Then $f(P_0) \subseteq P_0$, and the corresponding incidence matrix $\mathbf{A}$ is clearly not diagonalizable, with 1 as the only eigenvalue. In this case, it is easily calculated that $\mathrm{Var}(f^n) = n + 3$, and $1 - (\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n)) = (-1)/(n+3)$. Thus, even though $\log(\mathrm{Var}(f^{n+1})/\mathrm{Var}(f^n))$ converges to the entropy, the convergence of the error to zero is no longer exponential. It is not difficult to check that the "$+\mathrm{I}$" trick used above does not change this problem.

In the more general Markov case in which the incidence matrix is not diagonalizable, consider the Jordan normal form of the matrix. Then each term $c_i\lambda_i^n$ in the formula (*) above which has a nontrivial Jordan block is replaced by terms of the form $c_i\lambda_i^n + d_1(n)\,\lambda_i^{n-1} + \cdots + d_{k-1}\lambda_i^{n-k+1}$, where k is the size of the largest Jordan block for λ_i, and d_j is a polynomial of degree j. (This is easily seen by calculating the nth power of a typical Jordan block.)

If the Jordan blocks for the dominant eigenvalue are all trivial, then the situation is pretty much the same as for the diagonalizable case, and the Revised Algorithm will converge exponentially to the desired value.

If the dominent eigenvalue has nontrivial Jordan blocks, then it is easy to see that the revised algorithm still converges to the entropy in this case, but the convergence is no longer exponential, but of the form $O(1/n)$.

Given that every piecewise monotone map of the interval is arbitrarily close to a Markov map with the same number of monotone pieces, we have a clear heuristic reason for believing that this altered algorithm will always converge to the entropy. Of course, this observation does not constitute a proof of convergence of the revised algorithm in all cases, which is still an unsolved problem. Nevertheless, the above observations, along with the experimental data presented below, present a strong case that the Revised

Algorithm provides us with a useful method for calculating the entropy of piecewise monotone maps on the interval.

4. NUMERICAL RESULTS

We use the Revised Algorithm on the following one-parameter families of test functions, which are all maps of the unit interval to itself. Let s_n indicate the result of the Revised Algortithm after n iterations. Since the exact topological entropy as a function of the parameter is known only for family (1) below, it is difficult to devise an efficient test for the accuracy of the algorithm. We have given the values $|s_{201} - s_{200}|$ only as one possible indicator of the rate of convergence, although it is obvious that small values of $|s_{201} - s_{200}|$ do not necessarily imply that s_{200} is close to the actual entropy. For two of the families, we have also given the values of $|s_{51} - s_{50}|$. Our choice for the number of iterations given was based upon the limitations of the computational facilities which we had at hand.

1.

$$f(x) = \begin{cases} Ax, & 0 \leqslant x < 0.5 \\ A - Ax, & 0.5 \leqslant x \leqslant 1 \end{cases} \quad \text{where} \quad 0 \leqslant A \leqslant 1. \text{ (cf. [4].)}$$

2. $f(x) = Ax(1-x)$ where $3.5 \leqslant A \leqslant 4$. (cf. [4].)
3. $f(x) = x + A\sin(2\pi x)$ where $0.55 \leqslant A \leqslant 0.7326$. (cf. [3].)
4.

$$f(x) = \frac{x(x - A/2)^2}{(1 - A/2)^2} \quad \text{where} \quad 1.3 \leqslant A \leqslant 1.5. \text{ (cf. [3].)}$$

5. $f(x) = A(\sin(2\pi x) + 1)$ where $0.3 \leqslant A \leqslant 0.5$. (cf. [8].)
6. $f(x) = 0.5(1 + A\sin(4\pi x))$ where $0 \leqslant A \leqslant 1$.

These functions and the results of the algorithm are illustrated in Figs. 1–13 below. We use s_n to denote the result of the algorithm after the nth iteration.

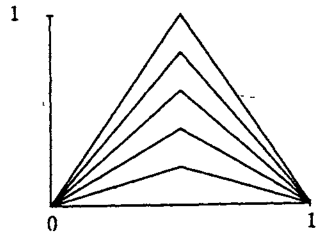


Fig. 1. The family $f(x) = Ax$ for $x \leqslant 0.5$ and $A - Ax$ for $0.5 < x \leqslant 1$ for $0 \leqslant A \leqslant 2$.

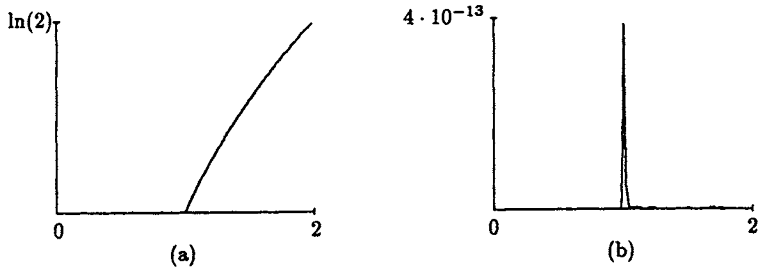


Fig. 2. (a) The topological entropy for the family $f(x) = Ax$ for $x \leqslant 0.5$ and $A - Ax$ for $0.5 < x \leqslant 1$ for 200 iterations and (b) the difference $|s_{201} - s_{200}|$.

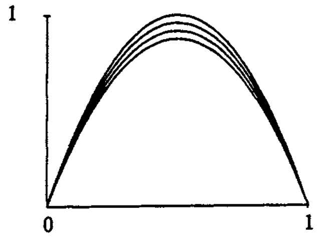


Fig. 3. The family $f(x) = Ax(1-x)$ for $3.5 \leqslant A \leqslant 4$.

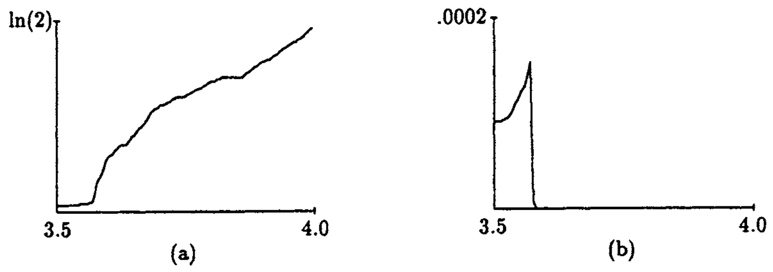


Fig. 4. (a) The topological entropy for the family $f(x) = Ax(1-x)$ for 200 iterations and (b) the difference $|s_{201} - s_{200}|$.

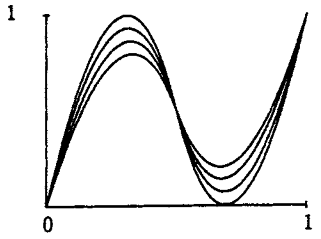
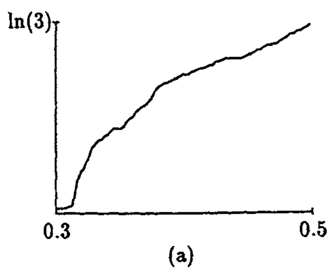


Fig. 5. The family $f(x) = x + A \sin(2\pi x)$ for $0.55 \leqslant A \leqslant 0.7326$.



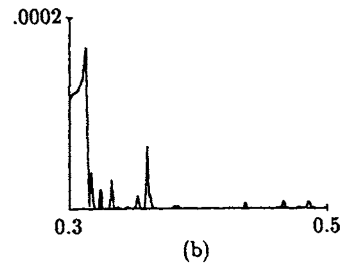


Fig. 6. (a) The topological entropy for the family $f(x) = x + A\sin(2\pi x)$ for 200 iterations and (b) the difference $|s_{201} - s_{200}|$.

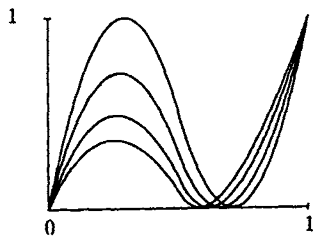
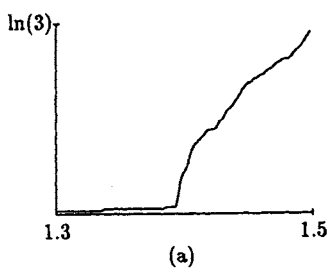


Fig. 7. The family $f(x) = x(x - A/2)^2/(1 - A/2)^2$ for $1.3 \leqslant A \leqslant 1.5$.



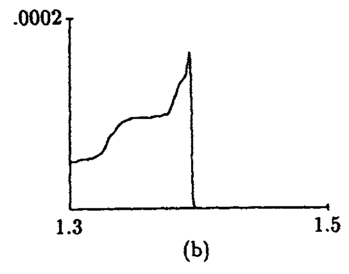


Fig. 8. (a) The topological entropy for the family $f(x) = x(x - A/2)^2/(1 - A/2)^2$ for 200 iterations and (b) the difference $|s_{201} - s_{200}|$.

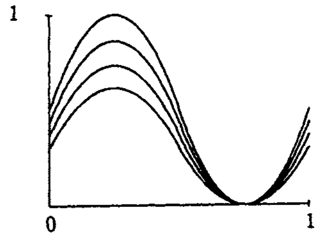


Fig. 9. The family $f(x) = A(\sin(2\pi x) + 1)$ for $0.3 \leqslant A \leqslant 0.5$.

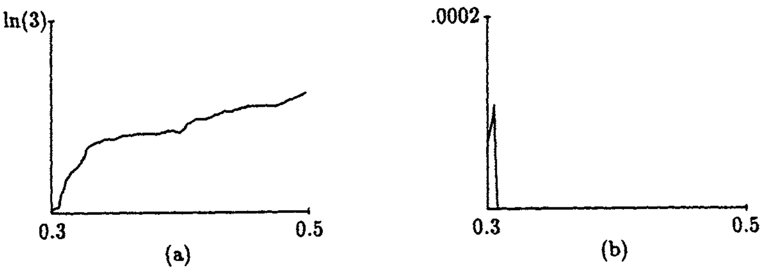


Fig. 10. (a) The topological entropy for the family $f(x) = A(\sin(2\pi x) + 1)$ for 200 iterations and (b) the difference $|s_{201} - s_{200}|$.

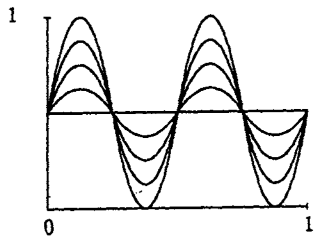


Fig. 11. The family $f(x) = \frac{1}{2} + (A/2)\sin(4\pi x)$ for $0 \leqslant A \leqslant 1$.

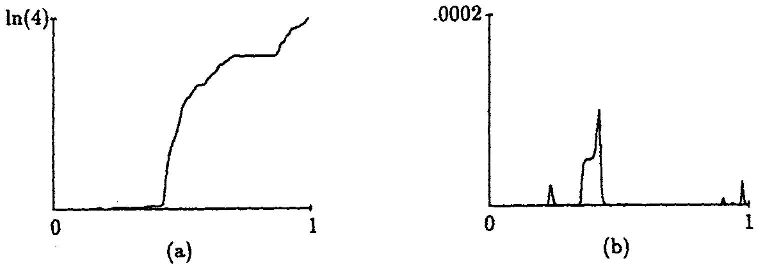


Fig. 12. (a) The topological entropy for the family $f(x) = \frac{1}{2} + (A/2)\sin(4\pi x)$ for 200 iterations and (b) the difference $|s_{201} - s_{200}|$.

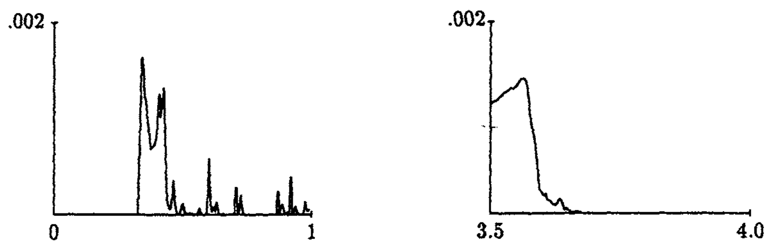


Fig. 13. The difference $|s_{51} - s_{50}|$ for functions 6. and 2. respectively. Compare with Figs. 12.b and 4.b.

We present below the results of the results of the Revised Algorithm for 50 iterations for Test Functions 1 and 3.

Though we can't know precisely what the error is for the Revised Algorithm, we can consider the value $|s_{n+1} - s_n|$. In the following table we present the highest such values obtained for the above functions.

Although the method appears to be an efficient way of calculating topological entropy, it must be acknowledged that the evidence for this is mainly experimental in the non-Markov case, and that the theoretical results needed to accurately assess the method (and its comparison to other methods) are still not available.

5. CONCLUSIONS

All of the algorithms under discussion have one important problem in common. They are not efficient when computing entropy close to zero (see the spikes and plateaus in Figs. 2.b, 4.b, 6.b, 8.b, 10.b and 11). This seems to be an inherent problem with algorithms intended to compute entropy and may be intractable. Since this also occurs for the family of tent maps, for which s_n ought to give the exact value for all n, it is clear that the entire problem, in this case, is due to roundoff error. In this paper we have not addressed the problem of round-off error for the Revised Algorithm. However, note that, in Example 1 (cf. Table I), the relative error for 200 iterations is much greater than that for 50 iterations.

Of the available algorithms, the Block–Keesling–Li–Peterson method (cf. [4]) is still the clear winner for those maps for which the method is valid, the unimodal maps. The same is largely true of the Block–Keesling method (cf. [3]) for bimodal maps which are not unimodal, although it is not as easily programmed as the unimodal algorithm (which can easily be programmed and debugged in a fraction of an hour).

Table I. The Relative Error for the Functions Discussed in the Text

Function	$\|s_{201} - s_{200}\|$	$\|s_{51} - s_{50}\|$
1.	3.870862×10^{-13}	5.141720×10^{-15}
2.	1.529862×10^{-4}	1.407561×10^{-3}
3.	1.689490×10^{-4}	5.752724×10^{-3}
4.	1.775111×10^{-4}	1.820075×10^{-3}
5.	1.081606×10^{-4}	1.044942×10^{-3}
6.	1.007171×10^{-4}	1.790196×10^{-3}

Of the algorithms mentioned above which work on piecewise monotone functions having more than three monotone pieces, only the algorithm of Balmforth, Spiegel and Tresser (cf. [2]), and the algorithm given here, appear to be practical. Both methods are motivated by the well-known Markov example, and are essentially equivalent for that case. Moreover, the strengths and weaknesses of these two algorithms appear to complement each other.

The Balmforth–Spiegel–Tresser algorithm has the strength that it offers calculations which give an upper and lower bound for $h(f)$. However, there does not appear to be any reliable method in the Balmforth–Spiegel–Tresser algorithm for deciding how fine the partition of the interval should be in order to get a certain accuracy. A partition must be decided in advance, and then most of the expense of the method is the time needed to estimate the eigenvalues of the corresponding matrices. If the choice is not accurate enough, then one must start over with a new partition and a new eigenvalue calculation.

In contrast, our method does not give such upper and lower bounds. On the other hand, the values s_n give a running estimate of the entropy, and in order to calculate s_{n+1}, one only needs two of the matrices which were used to calculate s_n plus a few simple additional steps. Such a running estimate is not feasible in the Balmforth–Spiegel–Tresser algorithm without a considerable investment in time to calculate the dominant eigenvalues of the corresponding matrices at each stage.

The contrast between these two methods suggests that our method might be combined with the Balmforth–Spiegel–Tresser method to get an algorithm which is better than either one alone. One possible way of combining the algorithms is to use our algorithm until $|s_{n+1} - s_n|$ is less than some function of the desired error, and then use the Balmforth–Spiegel–Tresser algorithm to get the actual upper and lower bounds for $h(f)$.

Since the method given here was partially motivated by a method for calculating eigenvalues, which was then translated to a setting appropriate for entropy, it is useful to ask what other eigenvalue methods might profitably be modified in order to get algorithms for calculating entropy.

REFERENCES

1. R. L. Adler, C. Konheim, and M. H. McAndrew, Topological entropy, *Trans. Amer. Math. Soc.* **114**:309–319 (1965).
2. N. J. Balmforth, E. A. Spiegel, and C. Tresser, Topological entropy of one-dimensional maps: Approximations and bounds, *Phys. Rev. Letters* **72**:80–83 (1994).
3. L. Block and J. Keesling, Computing the topological entropy of maps of the interval with three monotone pieces, *J. Stat. Phys.* **66**:755–774 (1992).

4. L. Block, J. Keesling, S. Li, and K. Peterson, An improved algorithm for computing topological entropy, *J. Stat. Phys.* **55**:929–939 (1989).
5. L. Block, J. Guckemheimer, M. Misiurewicz, and L. S. Young, Periodic Points and Topological Entropy of One Dimensional Maps in *Global Theory of Dynamical Systems* (Springer Lecture Notes in Mathematics, No. 819).
6. R. L. Burden and J. D. Faires, *Numerical Analysis*, Fifth Edition, PWS Publishing Company, Boston, 1993.
7. P. Collet, J. P. Crutchfield, and J.-P. Echmann, Computing the topological entropy of maps, *Commun. Math. Phys.* **88**:257–262 (1983).
8. P. Góya and A. Boyarsky, Computing the topological entropy of general one-dimensional maps, *Trans. Am. Math. Soc.* **323**:39–49 (1991).
9. R. A. Horn and C. R. Johnson, *Matrix Analysis* (Cambridge University Press, 1985).
10. J. Milnor and W. Thurston, On iterated maps of the interval, in *Lecture Notes in Mathematics*, No. 1342 (Springer-Verlag, 1988), pp. 465–563.
11. M. Misiurewicz, Horseshoes for mappings of the interval, *Bull. Acad. Polon. Sci. Ser. Sci. Math.* **27**:167–169 (1979).
12. M. Misiurewicz and W. Szlenk, Entropy of piecewise monotone mappings, *Studia Math.* **67**:45–63 (1980).
13. S. Newhouse and Thea Pignataro, On the estimation of topological entropy, *J. Stat. Phys.* **72**:1331–1351 (1993).
14. J. H. Wilkinson, *The Algebraic Eigenvalue Problem* (Oxford University Press, 1965).