The Algorithms Behind GAIO - Set Oriented Numerical Methods for Dynamical Systems

Michael Dellnitz*, Gary Froyland and Oliver Junge

Department of Mathematics and Computer Science
University of Paderborn
D-33095 Paderborn

Abstract. In a given dynamical system there are essentially two different types of information which could be of practical interest: on the one hand there is the need to describe the behavior of single trajectories in detail. This information is helpful for the analysis of transient behavior and also in the investigation of geometric properties of dynamical systems. On the other hand, if the underlying invariant set is generated by complicated dynamics then the computation of single trajectories may give misleading results. In this case there still exists important set related information covering both topological and statistical aspects of the underlying dynamical behavior. Within the DFG-Schwerpunkt we have focussed on the development of set oriented methods for the numerical approximation of

- invariant sets (e.g. invariant manifolds, global attractors, chain recurrent sets);
- (natural) invariant measures;
- almost invariant sets.

The basic concept is a subdivision algorithm which is similar in spirit to the well known cell mapping techniques but with the crucial difference that the numerical effort mainly depends on the complexity of the dynamics rather than on the dimension of the underlying state space. First, the invariant set is covered by boxes and then the dynamical behavior on the set is approximated by a Markov chain based on transition probabilities between elements of this covering. The algorithms have been implemented in the software package GAIO (**G**lobal **A**nalysis of **I**nvariant **O**bjects), and in this article we describe both the related numerical techniques together with their theoretical foundations and how to use them within GAIO. We will also discuss details concerning the implementation such as adaptive versions of the methods.

1 Introduction and Motivation

Our aim is to capture the global structure of a given dynamical system. This will be done by using global set-oriented methods rather than by an approach based on long term computations of single trajectories. At the topological level, objects of interest are foremostly invariant *sets*, as these sets support the dynamics of the system for all time. When the dynamics has some degree

* Research of the authors supported by the Deutsche Forschungsgemeinschaft under Grants De 448/5-1 – De 448/5-4.

of smoothness and hyperbolicity, invariant *manifolds* provide insight into the geometric structure of the system's dynamics. At the level of statistics, invariant *measures* quantitatively describe frequencies of visitation of trajectories to different regions of phase space. These invariant measures are generalised fixed eigenfunctions of a global *transfer operator*. Other eigenfunctions of this operator provide information such as identifying *almost invariant sets* which help to fine-tune the dynamical analysis. Based on the identification of these eigenfunctions the groups of Deuflhard and Schütte and co-workers have derived a new approach to the identification of conformations of biomolecules [8,30] within this DFG research program.

The above list gives a quick run-down of *what* our set-oriented methods set out to find. *How* we do this is arguably of even greater importance. We will begin each section with a concise introduction to the mathematical object under consideration. We then give a description of the algorithm used to approximate the object, followed by rigorous results concerning convergence. The theoretical exposition will be paralleled by examples to demonstrate the efficacy of the methods.

All of the algorithms described here have been coded in the software package GAIO (**G**lobal **A**nalysis of **I**nvariant **O**bjects). It is the existence of this software package that makes the use of our techniques feasible, allowing rapid computations and informative visualisations. Many figures in this paper have been made using visualisation techniques which have been developed and implemented within the software platform GRAPE by the group of Rumpf and co-workers [29,5,7] within the DANSE program. To highlight the ease of use of the software, we present actual GAIO commands applied to each of the instructive examples. These commands will illustrate the use of the package for real problems, and collectively will provide sufficient detail so as to act as a concise tutorial for it.

2 The Computation of Invariant Sets

Our setting is that of a continuous mapping $T : M \to M$ on a compact manifold M. We will call a set $A \subset M$ *forward invariant* if $T(A) \subset A$, *backward invariant* if $T^{-1}(A) \subset A$ and *invariant* if $T(A) = T^{-1}(A) = A$. There are various collections of invariant sets that one can talk about. The largest invariant set may be defined as follows.

Definition 1. Let $\mathcal{O}^{\pm}(x) = \{\dots, T^{-1}x, x, Tx, \dots\}$ denote the *full orbit* of a point $x \in M$. If T is non-invertible, then $T^{-k}x = \{y \in M : T^k y = x\}$. The *maximal invariant set* contained in a set $Q \subset M$ is defined as

$$\mathrm{Inv}(Q) = \{x \in Q : \mathcal{O}^{\pm}(x) \subset Q\} \tag{1}$$

It is straightforward (see e.g. [10]) to show the following properties of maximal invariant sets.

Proposition 2. *1.* Inv(Q) *is an invariant set.*
2. If $Y \subset Q$ is an invariant set, then $Y \subset \mathrm{Inv}(Q)$.
3. If Q is forward invariant, then $\mathrm{Inv}(Q) = \bigcap_{k=0}^{\infty} T^k(Q)$.
4. If T is a homeomorphism, then $\mathrm{Inv}(Q) = \bigcap_{k=-\infty}^{\infty} T^k(Q)$.

The maximal invariant set is defined setwise and so it contains many points which are not *recurrent* in the sense that under iteration they do not return close to themselves infinitely often. This leads to:

Definition 3. A point $q \in Q$ belongs to the *chain recurrent set* of T in Q if for every $\epsilon > 0$ there is an ϵ-pseudoperiodic orbit containing q, that is, there exists $\{q = q_0, q_1, \ldots, q_{\ell-1}\} \subset Q$ such that

$$\|T(q_i) - q_{i+1 \bmod \ell}\| \leq \epsilon \text{ for } i = 0, \ldots, \ell - 1.$$

Also the following result follows immediately from the definitions.

Proposition 4. *The chain recurrent set $R_Q(T)$ of T in Q is closed and invariant. Furthermore we have the following inclusion:*

$$R_Q(T) \subset \bigcap_{k \geq 0} T^k(Q). \tag{2}$$

Example 5. To illustrate the case where the inclusion of (2) is sharp, consider the map $T : [0, 1] \to [0, 1]$ defined by $Tx = x^2$. Here, $R_{[0,1]}(T) = \{0, 1\}$, while $\bigcap_{k \geq 0} T^k([0, 1]) = [0, 1]$.

This simple example illustrates the two qualitatively different types of invariant sets that we intend to approximate.

2.1 Algorithm: Relative Global Attractor

We describe numerical methods to approximate the invariant sets $\mathrm{Inv}(Q)$ and $R_Q(T)$. These methods are based on multilevel subdivision techniques. Let us begin with an abstract algorithm of this type for the computation of the *relative global attractor* $A_Q(T) = \bigcap_{k \geq 0} T^k(Q)$, where $Q \subset M$ is an arbitrary (not necessarily forward invariant) box, i.e. a generalized rectangle $B(c, r)$ with center c and radius r.

The idea of the algorithm is to cover $A_Q(T)$ by a finite number of boxes and to recursively tighten the covering by refining appropriately selected boxes.

Algorithm 1 (Subdivision Algorithm to Compute $A_Q(T)$). We start with the collection $\mathcal{B}_0 = \{Q\}$. For $k = 1, 2, \ldots$ compute $\mathcal{B}_k$ from the collection $\mathcal{B}_{k-1}$ in two steps:

1. *Subdivision:* Bisect each box in the current collection $\mathcal{B}_{k-1}$ into two smaller boxes of equal size (for d-dimensional boxes, the cutting plane is cycled around the d coordinate directions).

2. *Selection:* Discard those refined boxes whose preimage does not intersect any box of the current (refined) collection. The remaining boxes constitute the collection $\mathcal{B}_k$.

The following result is proved in [4].

Theorem 6. *Set* $Q_k = \bigcup_{B\in\mathcal{B}_k} B$ *and* $Q_\infty = \bigcap_{k\geq 0} Q_k$.

1. $A_Q(T) \subset Q_k$ *for all* $k \geq 0$
2. $T^{-1}Q_\infty \subset Q_\infty$
3. $A_Q(T) = Q_\infty$

Since $A_Q(T)$ is the set of all points which stay inside Q under backward iteration, it follows immediately that

$$\mathrm{Inv}(Q) = A_Q(T) \cap A_Q(T^{-1}),$$

whenever T is invertible.

Remark 7. 1. Results on the speed of convergence can be obtained if $A_Q(T)$ possesses a hyperbolic structure, see [4]. Roughly speaking the stronger the contraction along the stable direction the better is the convergence behaviour.
2. The algorithm has been adapted and successfully applied to the context of *random dynamical systems* [21] by Keller and Ochs within this DFG research program (project of L. Arnold).

2.2 Practicalities: Relative Global Attractor

In GAIO, the selection criterion in step (ii) of Algorithm 1 is tested using a set of test points in each box. These test points are mapped forward one step and if at least one of these image points lies inside the box B, then B is not discarded. This procedure can be made rigorous (in the sense that $A_Q(T)$ is always covered by the box collection) through knowledge of local Lipschitz constants of T [19]; see Appendix A.

To perform one step of the algorithm with GAIO, one simply types:

```
rga(tree)
```

(the mnemonic `rga` is a short hand for "**r**elative **g**lobal **a**ttractor"), where `tree` is the data structure containing the current collection of boxes. In fact the boxes are stored in a binary tree, where the children of a box at depth k are the two boxes of half-size at depth $k+1$ formed by dividing the box at depth k into two equal pieces. This binary tree structure allows for rapid searching of which box contains the images of test points, and reduces the time for the subdivision procedure from $O(n^2)$ to $O(n\log n)$, where n denotes the number of boxes. Most of the algorithms we describe in this paper benefit significantly from the tree structure.

2.3 Example: Relative Global Attractor of a Knotted Flow

Consider a flow of a three-dimensional ordinary differential equation through an open-ended cylinder from top to bottom. On the mantle of the cylinder, the flow proceeds directly downwards. Near the center of the cylinder, some trajectories of the flow loop around to form a knot; see the red portion of Figure 4. Using techniques from algebraic topology, it is possible to prove that there is a non-trivial invariant set contained inside the cylinder (see [7] for details). Obviously this invariant set is unstable, and therefore extremely difficult, if not impossible, to be observed numerically via simulation of single trajectories. Our set oriented approach is particularly suitable, as we cover the entire cylinder with coarse boxes and then repeatedly refine and discard boxes which are known not to contain a part of the invariant set.

In GAIO, one issues the commands

```
knot = Model('Knot')
rk4 = Integrator('RungeKutta4')
rk4.model = knot
rk4.h = 0.1
rk4.tFinal = 1.5

tree = Tree(knot.center, knot.radius)
tree.integrator = rk4
tree.domain_points = Points('Grid', knot.dim, 125)
tree.image_points = Points('Vertices', knot.dim)

rga(tree, 20)
```

The first block of commands loads the model file `Knot` into GAIO (see Appendix B on how to define your own model), and sets the parameters for the integration of the vector field defined in `Knot`. For instance here we use a fourth-order Runge-Kutta scheme in the numerical integration. The variable `rk4.h` determines the integration step-size, and `rk4.tFinal` indicates that we will treat the ODE as a 1.5-time discrete map (one iteration of the discrete map is defined by integration for 1.5 time units).

The second block of commands initialises the tree structure, and the set of test points that will be used in the selection step (ii) of Algorithm 1. Here we use a uniform grid of 125 test points (`domain_points`) in each box. Additionally we choose a set of `image_points` (given here by the vertices of a box); see Appendix A for an explanation on this.

We perform 20 subdivision steps on the initial collection given by the box with center `knot.center` and radius `knot.radius`. The covering of the corresponding backward invariant set consists of 267458 boxes; see Figures 1 and 2. This has to be viewed as an approximation of the unstable invariant set together with its unstable manifold.

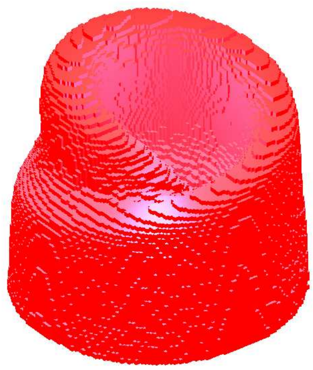

Fig. 1. Covering of the relative global attractor for the knotted flow.

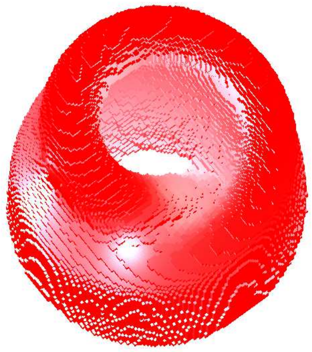

Fig. 2. As for Figure 1, from another viewpoint.

2.4 Algorithm: Chain Recurrent Sets

Often the maximal invariant set contains many "transient" points which we sometimes would like to eliminate. For instance, in Example 5, we would additionally like to know that the two points $x = 0$ and $x = 1$ are all of the recurrent points, rather than simply state the obvious fact that the entire interval $[0, 1]$ is invariant. We now show how to modify Algorithm 1 in such a way that we can approximate the chain recurrent set. As in Algorithm 1 we construct a sequence $\mathcal{B}_0, \mathcal{B}_1, \dots$ of collections of boxes creating successively tighter coverings of the desired object.

Algorithm 2 (Subdivision Algorithm to Compute $R_Q(T)$). Set $\mathcal{B}_0 = \{Q\}$. For $k = 1, 2, \dots$ the collection $\mathcal{B}_k$ is obtained from $\mathcal{B}_{k-1}$ in two steps:

1. *Subdivision:* Bisect each box in the current collection $\mathcal{B}_{k-1}$ into two smaller boxes of equal size (for d-dimensional boxes, the cutting plane is cycled around the d coordinate directions).
2. *Selection:* Construct a directed graph whose vertices are the boxes in the refined collection and by defining an edge from vertex B to vertex B', if

$$T(B) \cap B' \neq \emptyset. \tag{3}$$

   Compute the strongly connected components of this graph and discard all boxes which are not contained in one of these components.

Remark 8. Recall that a subset W of the nodes of a directed graph is called a *strongly connected component* of the graph, if for all $w, \tilde{w} \in W$ there is a path from w to $\tilde{w}$. The set of all strongly connected components of a given directed graph can be computed in linear time [26].

Intuitively it is plausible that the sequence of box coverings $\mathcal{B}_k$ converges to the chain recurrent set of T. Indeed, under mild assumptions on the box coverings one can prove convergence, see [11,27].

2.5 Example: Chain Recurrent Set of the Knotted Flow

We return to the previous example (the flow through an open-ended cylinder). The computations are prepared in the very same way as before. One step of Algorithm 2 is now performed by executing

```
crs(tree)
```

(where `crs` is meant to be an abbreviation for "**c**hain **r**ecurrent **s**et"). Here the `domain_points` and `image_points` of the tree are used to compute the directed graph in a way similar to the selection step of algorithm 1. The result after 18 subdivision steps is shown in Figure 3, where the approximate chain recurrent set is shown in blue (11567 boxes), overlaying the relative global attractor. Figure 4, which has been produced by Robert Strzodka, shows a covering of the chain recurrent set in dark blue after 30 subdivision steps.

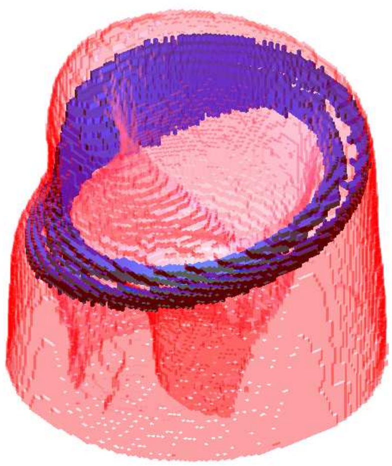

Fig. 3. Coverings of the relative global attractor (red), and the chain recurrent set (blue).

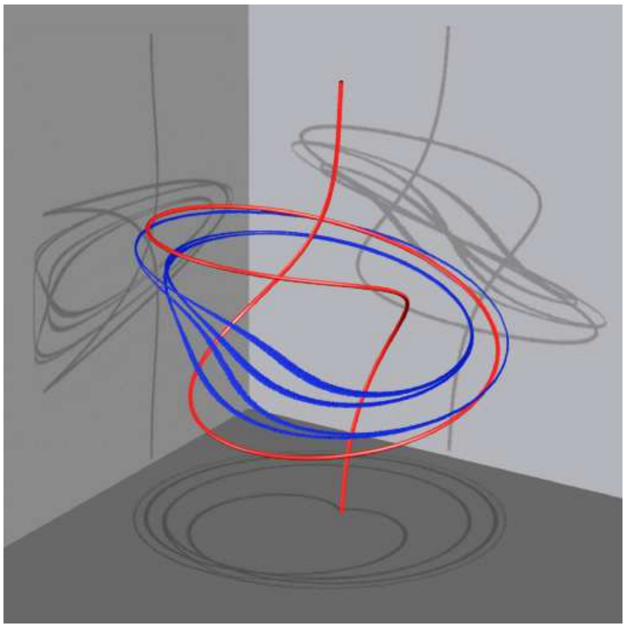

Fig. 4. Covering of the chain recurrent set (blue) at a deeper level of subdivision. The knot trajectory that defines the flow is shown in red.

3 Invariant Manifolds

The set oriented techniques may be applied to provide rigorous coverings of invariant manifolds within some prescribed box Q. For simplicity, we describe only the situation of unstable manifolds of hyperbolic fixed points of diffeomorphisms. However we emphasise that in principle the algorithm can be applied to general stable or unstable manifolds of arbitrary invariant sets.

Definition 9. Let x_0 be a hyperbolic fixed point for the diffeomorphism $T : M \to M$. Let U be a neighbourhood of x_0 and define the *local stable manifold* of x_0 by

$$W^s(x_0, U) = \{y : T^j y \in U \text{ for } j \in \mathbb{Z}^+ \text{ and } d(T^j y, x_0) \stackrel{j\to\infty}{\longrightarrow} 0\} \tag{4}$$

where $d(\cdot, \cdot)$ is a metric on M. The *local unstable manifold* is defined as

$$W^u(x_0, U) = \{y : T^{-j} y \in U \text{ for } j \in \mathbb{Z}^+ \text{ and } d(T^{-j} y, x_0) \stackrel{j\to\infty}{\longrightarrow} 0\}. \tag{5}$$

We have the following simplified version of the stable manifold theorem, see e.g. [28].

Theorem 10. *Let x_0 be a hyperbolic fixed point for the C^k diffeomorphism $T : M \to M$. Then there is a neighbourhood $U' \subset U$ such that the sets $W^s(U', x_0)$ and $W^u(U', x_0)$ are C^k embedded disks.*

The global stable *and* unstable manifolds *may be obtained by*

$$W^s(x_0) = \bigcup_{j \geq 0} T^{-j} W^s(x_0, U') \quad \text{and} \quad W^u(x_0) = \bigcup_{j \geq 0} T^j W^u(x_0, U'), \quad (6)$$

respectively.

3.1 Algorithm: Invariant Manifolds

The rough idea behind covering the unstable manifold is as follows. Firstly, use GAIO to identify small regions containing fixed points (assuming that the fixed points are not known *a priori*). This can be done via a cycle of subdividing and throwing away all boxes whose image does not intersect itself. Remaining boxes cover all fixed points.

Once a fixed point has been located with sufficient precision, we apply Algorithm 1 to a small box containing the fixed point. Beginning with the obtained collection of boxes, these are mapped forward one iteration, and the boxes that they "hit" are added to the collection. These new included boxes are then mapped forward, and the procedure is repeated. In this way we obtain a covering (which can be made rigorous using Lipschitz estimates on the map as before; see Appendix A) of part of the global unstable manifold. Formally, the algorithm consists of two main steps:

Algorithm 3 (Continuation Algorithm to Compute $W^u(x_0)$).

1. *Initialisation:* Apply Algorithm 1 to a small box containing the hyperbolic fixed point x_0. Let the resulting collection be part of a partition of Q. Repeat the following step until no more boxes are added to the current collection:
2. *Continuation:* Map the obtained collection of boxes forward and note which other boxes of the partition are hit by these images. Add these boxes to the collection.

Remark 11.

1. It can be shown that Algorithm 3 indeed converges to part of the unstable manifold. For a detailed description of this convergence result see [3].
2. Recently there have also been results obtained on the speed of convergence in case where the unstable manifold is contained in a hyperbolic attractor, see [18]. As expected like in the case of the Subdivision Algorithm 1 the speed of convergence crucially depends on the contraction rate along the stable direction.
3. Observe that in the realisation it is not necessary to partition Q a priori into small boxes. Rather we use the same hierarchical data structure as for Algorithm 1 and just add leaves to the tree when the corresponding boxes are hit.

4. In order to cover the stable manifolds, the continuation algorithm may be applied to the inverse map T^{-1}.

3.2 Practicalities

To initialise the computations in GAIO we construct a single small box B around the fixed point x_0 within the tree data structure:

```
tree.insert(x_0, depth)
```

Here `depth` specifies at which depth of the tree the box will be generated. A higher depth corresponds to a smaller box. We may then apply Algorithm 1 in order to obtain a covering of the local unstable manifold of x_0 in B:

```
steps = 6
rga(tree, steps)
```

Finally we prepare the current collection

```
inserted = 2
tree.set_flags('all', inserted)
```

and apply several steps of Algorithm 3:

```
steps = 4
gum(tree, tree.depth, steps)
```

(here the mnemonic `gum` abbreviates "**g**lobal **u**nstable **m**anifold"). The above `tree.set_flags` command (which just "marks" all boxes in the current collection with the flag "2") is necessary as a preparatory step since only newly inserted boxes are mapped forward in each step of the procedure `gum`.

3.3 Example: Computation of a Stable Manifold in the Lorenz System

We consider the problem of covering the two-dimensional stable manifold of the origin for the Lorenz system governed by the system of ODE's:

$$\begin{aligned}\dot{x} &= \sigma(y-x)\\ \dot{y} &= \rho x - y - xz\\ \dot{z} &= xy - \beta z\end{aligned}$$

with $\sigma = 10, \rho = 28$, and $\beta = 8/3$. Figure 5 was produced with the following commands:

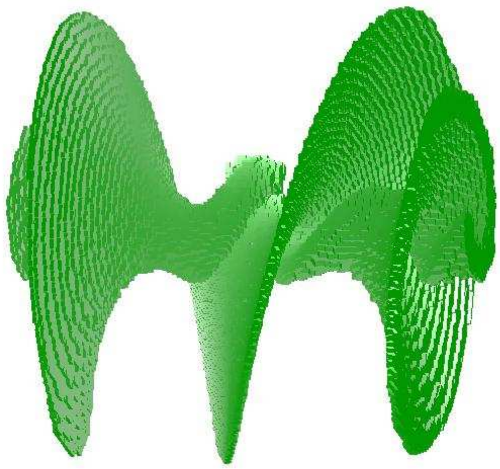

Fig. 5. Covering of the two-dimensional stable manifold of the steady state $(0, 0, 0)$.

```
lorenz = Model('lorenz')

rk4 = Integrator('RungeKutta4')
rk4.model = lorenz
rk4.tFinal = -0.1
rk4.h = -0.01

tree = Tree([0, 0, 0], [120, 120, 160])
tree.integrator = rk4
tree.domain_points = Points('Edges', 3, 100)
tree.image_points = Points('Center', 3)

x = [0; 0; 0]
depth = 21
tree.insert(x, depth)

steps = 10
gum(tree, depth, steps)
```

As usual we load the model, define the integration scheme and set up the tree object. Note that `rk4.tFinal = -0.1` and `rk4.h = -0.01`; so we integrate the ODE backwards in time. The unstable manifold of the timereversed system is equal to the stable manifold of the forward time system. In the second block of commands we insert the box containing the origin into the tree at depth 21. Then we apply 10 steps of the continuation algorithm. Note that here we do not need to issue the `tree.set_flags` command since we inserted a single box into the tree and did not perform the subdivision algorithm on this box.

3.4 Example: Rigorous Covering of an Unstable Manifold of the Hénon System

The algorithm for generating a rigorous covering of invariant manifolds will be illustrated with the Hénon mapping $T : \mathbb{R}^2 \to \mathbb{R}^2$, given by $T(x, y) = (1 - ax^2 + y, bx)$, with $a = 1.0$ and $b = 0.54$. Using the outer box $Q = [-2.2, 3.8] \times [-2.6, 3.4]$ the lightly shaded boxes in Figure 6 were produced by the following commands:

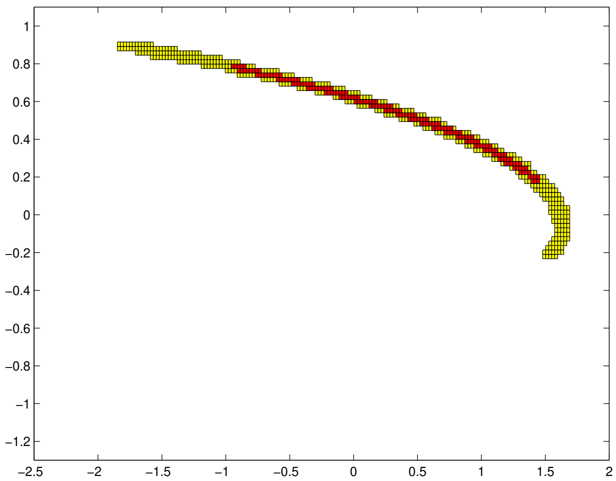

Fig. 6. Rigorous (light) and non-rigorous (dark) continuation applied to cover the unstable manifold of one of the fixed points of the Hénon map (6 continuation steps).

```
henon = Model('henon')
henon.a = 1.0
henon.b = 0.54

map = Integrator('Map')
map.model = henon
```

```
tree = Tree(henon.center, henon.radius)
tree.integrator = map
tree.domain_points = Points('Lipschitz', 2)
tree.image_points = Points('Vertices', 2)

depth = 16
x = henon.fixed_point
tree.insert(x, depth)

steps = 6
gum(tree, depth, steps)
```

Note the use of the point styles `Lipschitz` and `Vertices`; this option tells GAIO that the rigorous box intersection procedure (see Appendix A) is to be used. We insert the fixed point of the Hénon map into the tree at depth 16 and perform six continuation steps to extend the manifold.

The dark boxes in Figure 6 were produced by repeating the commands given above, but replacing the commands

```
tree.domain_points = Points('Lipschitz', 2)
tree.image_points = Points('Vertices', 2)
```

by

```
tree.domain_points = Points('Edges', 2, 100)
tree.image_points = Points('Center', 2)
```

yielding a non-rigorous computation of box intersections. Figure 7 shows both manifold coverings (rigorous and non-rigorous) extended by a further three steps, making a total of nine continuation steps. Finally, we perform the rigorous continuation method at depth 24 for a total of 19 steps; the result is shown in Figure 8.

4 Invariant Measures

An invariant measure describes the distribution of points on long trajectories, with regions that are visited more often being given higher "weight" or measure. Deterministic dynamical systems typically support many invariant measures. Under mild conditions on the dynamical system, it may be shown that by adding smooth localised dynamical noise, the resulting system has a unique invariant measure. Numerically, this "noisy" measure appears to be similar to the distribution of long trajectories for the original system for a large set of initial points in a neighbourhood of the chain recurrent set. In fact there are results [23] that prove that this is true for certain types of noise added to uniformly hyperbolic diffeomorphisms.

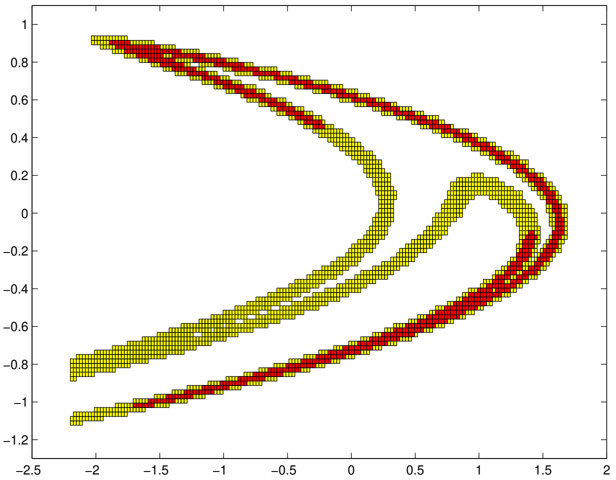

Fig. 7. Rigorous (light) and non-rigorous (dark) continuation applied to cover the unstable manifold of one of the fixed points of the Hénon map (9 continuation steps).

Definition 12. A probability measure μ on M is called T*-invariant* if $\mu \circ T^{-1} = \mu$. We are particularly interested in the situation where there is an invariant measure μ with the property that

$$\#\{0 \le k \le N-1 : T^k x \in A\}/N \to \mu(A) \qquad \text{as } N \to \infty \tag{7}$$

for every measurable $A \subset M$ and for Lebesgue almost all x in a neighbourhood of the chain recurrent set. Such an (obviously unique) invariant measure will be called a *physical measure* or *natural invariant measure* for T.

One iteration of our noisy system will be an application of T followed by a small perturbation; that is, $x \mapsto Tx + e$, where $e \in \mathbb{R}^d$ is small and of order ϵ. We formalise this by considering the noisy process to be a Markov chain. This Markov chain is completely defined by a transition function $\mathbf{Q}(\cdot,\cdot) : M \times \mathcal{B}(M) \to [0,1]$, where $\mathcal{B}(M)$ denotes the collection of Borel sets on M.

Example 13. 1. Let $B(x,\epsilon)$ denote the density of the uniform distribution restricted to $B_\epsilon(x)$, an ϵ ball about x. If $x \mapsto Tx + e$ with e selected from the density $B(0,\epsilon)$, then the corresponding transition function is $\mathbf{Q}_\epsilon(x,A) = \int_A B(Tx,\epsilon)\, dm(x)$, where m is Lebesgue measure. Similarly if $B(x,\epsilon)$ is replaced by the multidimensional Gaussian distribution $G(x,\epsilon)$ with mean x and variance ϵ then the corresponding transition function is $\mathbf{Q}_\epsilon(x,A) = \int_A G(Tx,\epsilon)\, dm(x)$.

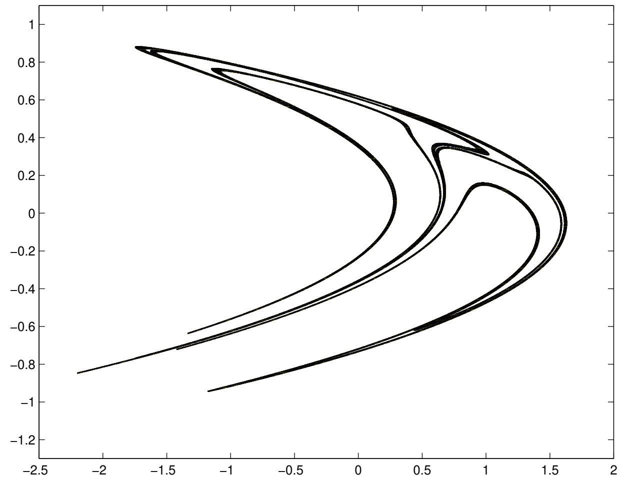

Fig. 8. The rigorous continuation algorithm applied to cover the unstable manifold of a fixed point of the Hénon map: 19 continuation steps at depth 24.

2. The transition function $\mathbf{Q}(x, A) = 1$ if $Tx \in A$ and $\mathbf{Q}(x, A) = 0$ otherwise, describes the (deterministic) Markov chain corresponding to the unperturbed map T.

Definition 14. Let $\mathcal{M}(M)$ denote the space of Borel probability measures on M. Given a transition function $\mathbf{Q}_\epsilon$, we may define a linear operator on $\mathcal{M}(M)$ that describes how probability measures are "pushed forward" under one step of the Markov chain. Define

$$(\mathcal{P}_\epsilon \nu)(A) = \int_M \mathbf{Q}_\epsilon(x, A)\, d\nu(x) \tag{8}$$

where $\nu \in \mathcal{M}(M)$. In cases where we use the deterministic transition function $\mathbf{Q}$ of Example 13 (ii), we denote the deterministic operator by $\mathcal{P}$ and call it the *Perron-Frobenius operator*. The operators $\mathcal{P}_\epsilon$ will be called *noisy Perron-Frobenius operators.*

Theorem 15 ([24]). *Suppose that for all $x \in M$, $\mathbf{Q}_\epsilon(x, \cdot)$ is an absolutely continuous probability measure. Let $\mathbf{Q}_\epsilon^{(k)}$ denote the transition function for k steps of the Markov chain. If, additionally, there exists a k_0 such that the probability measure $\int_M \mathbf{Q}_\epsilon^{(k_0)}(x, \cdot)\, dm(x)$ has a strictly positive density, then $\mathcal{P}_\epsilon$ has a unique fixed point, denoted μ_ϵ. Furthermore μ_ϵ is an everywhere positive absolutely continuous probability measure, and is the unique invariant measure for the noisy system.*

Remark 16. As an example, one can choose

$$\mathbf{Q}_\epsilon(x, A) = \mathcal{N}(x) \int_A \exp(-\|Tx - y\|^2/2\epsilon)\ dm(y),$$

where $\mathcal{N}(x)$ is a normalising factor. Such a class of perturbations are considered in [31], and this provides a very readable introduction to this "noising-up" approach. In this case, $k_0 = 1$ in the above theorem.

Theorem 15 holds for very general transition functions $\mathbf{Q}_\epsilon$; they do not need to be connected in any way to a deterministic mapping T. However, we are implicitly considering only those $\mathbf{Q}_\epsilon$ which define Markov chains whose dynamics is "close" to that of T. We now make this precise.

Definition 17 ([22,23]). We will say that a family of transition functions $\{\mathbf{Q}_\epsilon\}_{\epsilon>0}$ represents a *small random perturbation* of a continuous map T if

$$\mathbf{Q}_\epsilon(x, \cdot) \to \delta_{T(x)} \quad \text{weakly as } \epsilon \to 0 \tag{9}$$

uniformly in x. Here δ_y denotes the Dirac measure at y.

Theorem 18 ([22,23]). *Suppose that $\mathbf{Q}_\epsilon$ represents a small random perturbation of T and satisfies the conditions of Theorem 15. Let $\tilde{\mu}$ be a weak limit of $\{\mu_\epsilon\}$ as $\epsilon \to 0$, where a subsequence is selected if necessary. Then $\tilde{\mu}$ is T-invariant.*

In the case where $\mathbf{Q}_\epsilon(x, \cdot)$ is absolutely continuous for all x our noisy Perron-Frobenius operators may be considered as operators on L^1. They map the space of *densities* $D = \{f \in L^1 : f \geq 0, \int_M f\ dm = 1\}$ into itself. One may think of D as representing all absolutely continuous probability measures. For technical reasons, it is often advantageous for $\mathcal{P}_\epsilon$ to be a compact operator on $L^1(M, m)$, the space of integrable functions on M. Under some further mild conditions on $\mathbf{Q}_\epsilon$, this is also true.

Theorem 19 ([18]). *Suppose that $\mathbf{Q}_\epsilon(x, \cdot)$ has a Lipschitz density for all $x \in M$. Then $\mathcal{P}_\epsilon$ is compact as an operator on L^1.*

4.1 Algorithm: Natural Invariant Measures

The box coverings introduced earlier will form the backbone of our finite-dimensional approximation of the infinite-dimensional operator $\mathcal{P}$. More precisely, for a given box collection $\{B_1, \ldots, B_n\}$ we form the (column stochastic) transition matrix

$$P_{ij} = \frac{m(B_j \cap T^{-1}B_i)}{m(B_j)}, \tag{10}$$

$i, j = 1, \ldots, n$. One computes the (assumed, unique) fixed right eigenvector p of $P = (P_{ij})$, representing the invariant distribution for the finite-state Markov chain defined by P. An approximate invariant measure μ_n is defined by assigning $\mu_n(B_i) = p_i$.

Algorithm 4 (Computation of Natural Invariant Measures).
1. Compute the matrix P above.
2. Find the (right) Perron eigenvector p of P.
3. Set $\mu_n(B_i) = p_i$, $i = 1, \ldots, n$.

In general for the deterministic case it is not clear whether or not this measure μ_n is a good approximation of the physical measure. However under certain assumptions it can be shown that the measures μ_n indeed converge to a natural invariant measure, see e.g. [25,13,14,9]. Moreover the following convergence result for the stochastically perturbed context has been shown in [6]. Here we denote by μ_n^ϵ the approximate invariant measure obtained by computing the Perron eigenvector of the transition matrix for the stochastically perturbed system.

Theorem 20. *Suppose that the diffeomorphism T has a hyperbolic attractor Λ, and that there exists an open set $U_\Lambda \supset \Lambda$ such that for the densities q_ϵ of the transition functions $\mathbf{Q}_\epsilon(x, A) = \int_A q_\epsilon(Tx, y)\, dm(y)$ we have*

$$q_\epsilon(x, y) = 0 \quad \text{if } x \in \overline{T(U_\Lambda)} \text{ and } y \notin U_\Lambda.$$

Then the transition function $\mathbf{Q}_\epsilon$ has a unique invariant measure μ_ϵ with support on Λ and the approximating measures μ_n^ϵ converge to the natural measure μ of T as $\epsilon \to 0$ and $n \to \infty$,

$$\lim_{\epsilon \to 0} \lim_{n \to \infty} \mu_n^\epsilon = \mu.$$

4.2 Practicalities

The crucial algorithmic step is to compute the transition probabilities between boxes, i.e. the entries of the matrix P. In GAIO, this is carried out in two ways:

Computation of P Using Test Points The first method is to select a collection of m test points within each box. The points $\{x_1, \ldots, x_m\} \in B_j$ are mapped forward by T and we set

$$P_{ij} = \#\{x \in \{x_1, \ldots, x_m\} : Tx \in B_i\}/m.$$

For example, to use a set of m points distributed randomly according to a uniform distribution in each box one would use:

```
mc = Points('MonteCarlo', dim, m)
P = tree.matrix(mc, depth)
```

where `dim` denotes the dimension of phase space and `depth` specifies on which depth of the tree the transition matrix is to be computed. The variable `depth` may also be set to -1 in which case the transition matrix is computed on the leaves of the tree. This is useful in situations where we use adapted partitions (i.e. not all of the boxes are of the same size).

Computation of P Using an Exhaustion Technique The second method is to use an approach called "exhaustion", similar to the exhaustion techniques pioneered by Eudoxus [12]. To estimate the d-dimensional volume of $B_j \cap T^{-1}B_i$, the box B_j is repeatedly subdivided into smaller boxes until the forward image of a smaller box is known to fit completely inside B_i. This criterion is tested on the basis of Lipschitz estimates on the right hand side of the underlying model. At this point, subdivision of the sub-box stops. We also stop subdividing the sub-box when its volume has decreased beyond a certain threshold. A complete description of this method can be found in [16].

In GAIO, the transition matrix is created via the command:

```
P = tree.matrix('exhaustion', depth [, err])
```

In this command, the optional integer `err` is related to the volume threshold mentioned before. The subdivision of a sub-box stops, when its volume is smaller than $2^{-\texttt{err}}$ times the volume of B_j. The default value of `err` is 16.

Computing Eigenvalues and Eigenvectors of P To compute the fixed right eigenvector of the transition matrix P in GAIO, one types:

```
[v, l] = eigs(P, 1)
```

and the variable `l` will contain the largest real eigenvalue of P, with `v` containing the corresponding eigenvector.

Adaptive Partitioning Schemes So far, we have described how to compute transition matrices on a certain depth of the tree. Usually the corresponding collection will have been obtained by one of the algorithms described so far. They always lead to coverings with boxes of equal size. However, using information from the approximate invariant measure, it is possible to produce more efficient partitioning schemes. There are different strategies of how to use the information from the invariant measure, however, the basic algorithm has the following structure:

Algorithm 5 (Adaptive Subdivision Algorithm). From a box collection $\mathcal{B}_{k-1}$ and a corresponding approximate invariant measure μ_{k-1} compute $\mathcal{B}_k$ and μ_k in two steps:

1. *Subdivision*: Based on information from the approximate invariant measure, identify boxes which should be subdivided. Bisect each of those boxes into two smaller boxes of equal size (for d-dimensional boxes, the cutting plane is cycled around the d coordinate directions).
2. *Selection*: Compute the transition matrix and approximate the invariant measure μ_k for the refined box collection. Discard boxes for which the approximate measure is zero. The resulting collection constitutes $\mathcal{B}_k$.

We are now going to explain different identification procedures for step 1 of Algorithm 5 which are available in GAIO. For example one may

(a) subdivide boxes B for which $\mu_{k-1}(B) > 1/n$, where n is the number of boxes in $\mathcal{B}_{k-1}$;
(b) estimate a local approximation error from the (piecewise constant) density of the approximate invariant measure. Refine those boxes for which this estimated local error exceeds its average over all boxes (see [17] for a description of this approach).

The convergence of these adaptive schemes is analysed in detail in [18].

In GAIO, adaptive algorithm (a) is accessible via the commands

```
aim_hm(tree, method)
```

which performs one step of Algorithm 5 using subdivision procedure (a). The variable `method` determines which of the two above described methods is used in order to computed the transition matrices: it can be either a points object or the string `'exhaustion'`.

```
aim_lip(tree, method)
```

implements subdivision procedure (b). Note that for this approach to make sense we need a rather accurate result for the approximate invariant density, so in most cases one would exclusively choose the `method` to be `'exhaustion'` here.

4.3 Example: Bouncing Ball

We consider a discrete dynamical system that models a ball bouncing on a sinusoidally forced table. The approximate equations of motion are given by:

$$
\begin{aligned}
\phi_{t+1} &= \phi_t + v_t, \\
v_{t+1} &= \alpha v_t - \gamma \cos(\phi_t + v_t),
\end{aligned}
$$

with $\alpha = 0.9$ and $\gamma = 16$. Here $\phi_t \in [0, 2\pi)$ denotes the phase of the table at impact $\#t$, and $v_t \in \mathbb{R}$ denotes the exit velocity of the ball at impact $\#t$. Figure 9 shows a plot of a numerical orbit of length 10^5 with initial conditions $\phi_0 = 0$, $v_0 = 0$; that is, the points $\{(\phi_t, v_t)\}_{t=0}^{10^5}$ have been plotted on the phase space $M = S^1 \times [-100, 100]$.

Darker regions of Figure 9 contain more points, and therefore are visited more frequently by the numerical trajectory than lighter regions. Our estimate of the physical invariant measure will approximate the long term distribution of points in Figure 9 that arises in the $t \to \infty$ limit. Figure 10 is a gray scale plot of the density of an approximate invariant measure computed using the techniques described above.

Figure 11 shows the graph of this density in three dimensions. It can clearly be seen that this invariant density has a certain spatial structure

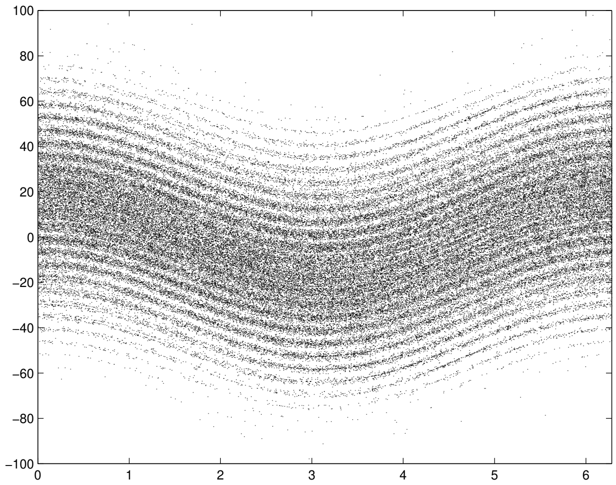

Fig. 9. A numerical trajectory of length 10^5 for the bouncing ball system.

which is reminiscent of the *symmetry on average* of attractors as described in [2].

To compute this approximation, we issued the following commands:

```
bb = Model('bouncingball')

map = Integrator('Map')
map.model = bb

tree = Tree(bb.center, bb.radius)
tree.integrator = map

to_be_subdivided = 8
for i=1:15
  tree.set_flags('all', to_be_subdivided)
  tree.subdivide(to_be_subdivided)
end

ig = Points('InnerGrid', 2, 1000)
P = tree.matrix(ig)
[v, l] = eigs(P, 1)
```

Note that because we subdivide *all* boxes at each step, we do not set the domain and image points in the first block (we don't need to worry about computing intersections of boxes). The decision to subdivide all boxes is only

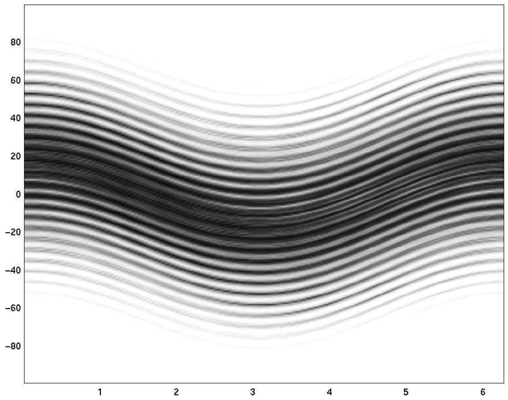

Fig. 10. Density of an approximate invariant measure using 65536 boxes (16 subdivisions). Darker areas correspond to higher density.

a little inefficient (from the data storage point of view) in this instance, as the positive density region occupies most of the phase space. We chose to subdivide all boxes purely because it is then easier to use the MATLAB visualisation function used to produce Figure 11. Normally instead of subdividing all boxes we would use one of the Subdivision Algorithms 1, 2 or 5.

In the for-loop we repeatedly mark all boxes with the flag "8" and then subdivide all boxes which have this flag set (i.e. all boxes are subdivded). Once the box collection at depth 16 has been produced by the for-loop, we set the variable `ig` to determine how to select test points for the computation of the transition matrix P. In this example, we use 1000 points per box, arranged in a uniform grid (set slightly away from the boundary of the boxes, as indicated by `InnerGrid`). We then generate the $2^{16} \times 2^{16}$ sparse matrix P and compute its Perron eigenvector as the approximate invariant measure.

5 Almost Invariant Sets and the Isolated Spectrum

Often one observes that in transitive systems, there are regions in which orbits stay for very long times before moving to other regions, only to return some longer time later. A well-known example are the two "wings" of the Lorenz attractor. Trajectories tend to stay on each wing for quite a long time before switching to the other wing. It is not just purely of dynamical interest to identify such "almost invariant sets"; the concept of "almost invariance" has recently also successfully been used for the identification of conformations for molecules (see [8,30]).

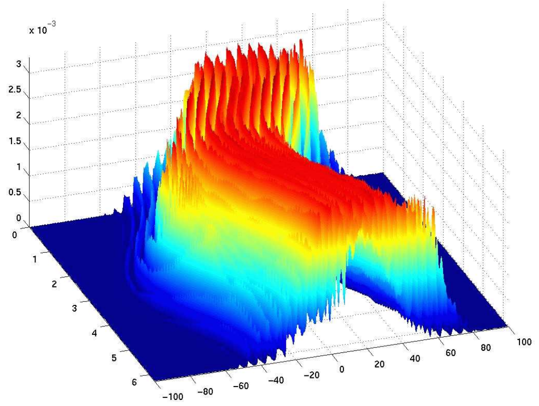

Fig. 11. A three-dimensional version of Figure 10 with the density plotted in the z-coordinate, and coloured according to the density.

We return to the operator $\mathcal{P}$ and consider its action on $\mathcal{M}_{\mathbb{C}}(M)$, the space of (possibly complex-valued) Borel measures (recall that $\mathcal{P}$ is defined by equation (8) using $\mathbf{Q}$ defined in Example 13 (ii)). Suppose that there is a real eigenvalue $0 < \lambda < 1$, with corresponding real eigenmeasure $\nu \in \mathcal{M}_{\mathbb{C}}(M)$ such that $\mathcal{P}\nu = \lambda\nu$. We assume that ν is normalised so that $|\nu|(M) = 1$ (this means that there are two disjoint subsets A_1, A_2 such that $\nu(A_1) = 1/2$, $\nu(A_2) = -1/2$, and $A_1 \cup A_2 = M$). We will say that the two disjoint subsets A_1, A_2 partition M into two *almost-invariant sets*. The following result [6] lends weight to this assertion.

Theorem 21. *Define*

$$\delta = \nu(A_1 \cap T^{-1}A_1)/\nu(A_1) \quad \textit{and} \quad \sigma = \nu(A_2 \cap T^{-1}A_2)/\nu(A_2).$$

Then $\delta + \sigma = \lambda + 1$.

The numbers δ and σ represent the amount of ν-mass that stays inside A_1 and A_2, respectively, under one iteration of T. Thus, if A_1 and A_2 are close to being invariant, δ and σ should be both close to one; this implies that λ is close to one also. If almost all of the ν-mass leaves A_1 in one iteration (and likewise for A_2), then both δ and σ should be close to zero; and λ should be close to -1. Thus λ close to -1 suggests that the two sets A_1 and A_2 form part of an *almost two-cycle*. Further generalisations are possible, and identical results hold for the noisy operator $\mathcal{P}_\epsilon$, see [6,15] for a detailed description.

5.1 Algorithm: Almost Invariant Sets

The transition matrix P constructed in the previous section provides a discrete approximation of the smooth dynamics. For motivational purposes let us suppose that the transition matrix is reducible, in the sense that there are two invariant subspaces V_1 and V_2 of dimension $n-r$ and r respectively, and that P restricted two each of V_1 and V_2 is irreducible. The eigenvalue one for the matrix P has geometric multiplicity two, and the spaces V_1 and V_2 can be extracted from the two fixed vectors of P (by using information on the positive and negative parts of the eigenvectors); see [6].

Suppose now that the system is perturbed, resulting in one of the eigenvalues moving away from one (one of the eigenvalues must stay at unity because the matrix remains stochastic). Then the matrix P becomes irreducible, but still close to being reducible. By continuity, the positive and negative parts of the eigenvectors corresponding to the second eigenvalue (the eigenvalue that moved away from unity) will approximate invariant sets. We call these *almost-invariant sets.*

To search for almost invariant sets, we look for eigenvalues of $\mathcal{P}$ (or $\mathcal{P}_\epsilon$) close to unity (or close to -1 when looking for almost two-cycles). We assume that the eigenmeasures and eigenvalues of $\mathcal{P}$ are well-approximated by eigenvectors and eigenvalues of the matrix P; at least for eigenvalues close to the unit circle. Such good approximation has been made precise for the noiseless operator $\mathcal{P}$ in one-dimension [20] and also for the compact operator $\mathcal{P}_\epsilon$ in higher dimensions [6]. Moreover, the existence of isolated eigenvalues and their dynamical relevance has recently been analytically studied for a certain class of one-dimensional maps, [1].

The algorithm for the computation of almost invariant sets is summarised below in the case where the second largest eigenvalue of P (in magnitude) is positive and real. The case of a large negative eigenvalue – leading to an almost invariant two-cycle – can be treated analogously.

Algorithm 6 (Computation of Almost Invariant Sets).

1. Compute the eigenvector v corresponding to the second largest real eigenvalue of P.
2. Create two index sets $\mathcal{I}_1 = \{i \in \{1,\dots,n\} : v_i \geq 0\}$ and $\mathcal{I}_2 = \{i \in \{1,\dots,n\} : v_i < 0\}$.
3. Denote the i^{th} box by B_i. The box collections $\widetilde{A}_1 := \bigcup_{i\in\mathcal{I}_1} B_i$ and $\widetilde{A}_2 := \bigcup_{i\in\mathcal{I}_2} B_i$ approximate A_1 and A_2 in Theorem 21.

Note that the construction of $\mathcal{I}_1$ and $\mathcal{I}_2$ in step 2 is somewhat arbitrary, see [15] for a more detailed exposition on this.

5.2 Practicalities

In GAIO, we calculate the matrix P as described in §4.2. To find the s largest (in absolute value) eigenvalues, we type

```
[v, lambda] = eigs(P, s)
```

5.3 Example: Chua's Circuit

We consider the set of differential equations

$$\begin{aligned}\dot{x} &= \alpha(y - m_0 x - (1/3)m_1 x^3)\\ \dot{y} &= x - y + z\\ \dot{z} &= -\beta y\end{aligned}$$

with the parameter values $\alpha = 16, \beta = 33, m_0 = -0.2$, and $m_1 = 0.01$.

A covering of the unstable manifold of one of the fixed points is first computed using the continuation algorithm. There is numerical evidence that this leads to a covering of an attracting set of the underlying ordinary differential equation.

```
chua = Model('chua')

rk4 = Integrator('RungeKutta4')
rk4.model = chua

tree = Tree(chua.center, chua.radius)
tree.integrator = rk4
tree.domain_points = Points('Edges', 3, 100)
tree.image_points = Points('Center', 3)

depth = 21
x = chua.fixed_point
tree.insert(x, depth)

steps = 100
gum(tree, depth, steps)
```

We load the model, set the integrator to be used, and define the points to be used in the intersection tests. We then insert a single box containing one of the fixed points into the tree data structure at depth 21, and apply the continuation algorithm for 100 steps (at most; the routine `gum` will stop if no more boxes are added to the collection).

We compute the transition matrix

```
ig = Points('InnerGrid', 3, 1000)
P = tree.matrix(ig)
```

and using the `eigs` command described above, we compute the leading eigenvalues of P, and find there is a positive, real eigenvalue close to one. The corresponding eigenvector v of this second largest eigenvalue has positive and negative parts and $\sum_i v_i = 0$; we consider v as a discrete approximation of the signed measure ν introduced earlier. We choose zero as a "separator", and consider the two sets of indices $\mathcal{I}_1 = \{i \in \{1, \ldots, n\} : v_i \geq 0\}$ and

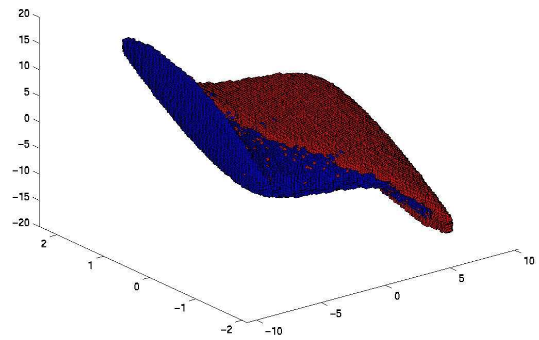

Fig. 12. Separation of the invariant set into two almost-invariant pieces.

$\mathcal{I}_2 = \{i \in \{1, \dots, n\} : v_i < 0\}$ to partition our box-covering into two almost invariant pieces, approximating the two sets A_1 and A_2. It can be numerically verified that the two sets of indices I_1 and I_2 satisfy a probabilistic version of Theorem 21 (replacing the action of T with the matrix P, and the eigenmeasure ν with the eigenvector v). These two sets are shown in red and blue, respectively, in Figure 12.

6 Acknowledgements

The authors would like to thank Michaela Schlör, Stefan Sertl, and Bianca Thiere for assistance in creating the example figures.

They gratefully acknowledge Kathrin Padberg's assistance in implementing the MATLAB interface to GAIO.

Figures 1-5 have been produced using the software platform GRAPE.

A Rigorous Calculation of Box Intersections

Because a finite number of test points is used to compute intersections of sets in Algorithms 1, 2 and 3, it is possible to miss some intersections, and therefore possibly not have a complete covering of an invariant set or manifold. By using information on the Lipschitz constants of the mapping (for simplicity we describe only the discrete-time case), it is possible to produce a rigorous covering of an invariant set or manifold, in the sense that the invariant object is completely contained inside the resulting collection of boxes.

The problem is as follows: Given a box B in a collection $\mathcal{B}$, find all boxes in $\mathcal{B}$ that intersect $T(B)$. Roughly speaking, the solution is this: Choose a finite grid of test points $\{x_1, \ldots, x_q\}$ in B in such a way that the distance between the images $T(x_i)$ and $T(x_j)$ of two neighbouring points x_i and x_j is less than the diameter of the boxes in $\mathcal{B}$. The set of all boxes in $\mathcal{B}$ intersected by the union of box-sized neighbourhoods centered at each $T(x_i)$ $(i = 1, \ldots, q)$ is then guaranteed to contain the collection of all boxes intersecting $T(B)$.

In order to impose an upper bound on the distance between $T(x_i)$ and $T(x_j)$, one needs information on the Lipschitz constants of T restricted to B. Let $r \in \mathbb{R}^d$ denote the radius of the boxes in $\mathcal{B}$ (recall that all boxes are of the same size). We need to know a $d \times d$ matrix L such that $|T(x) - T(y)| \leq L|x-y|$ for $x, y \in B$. If T is differentiable, then $L_{ij} = \max_{\xi \in B} |\partial_j T_i(\xi)|$ does the job, where T_i is the i^{th} component map of T. We now arrange test points in B lying on a d-dimensional grid $G = \{x \in B : x_i - c_i \in h_i \mathbb{Z}, i = 1, \ldots, d\}$, where $c = (c_1, \ldots, c_d)$ denotes the center of the box B and the vector $h \in \mathbb{R}^d$, $h > 0$, satisfies $Lh \leq 2r$.

Proposition 22. *Let* $\mathbf{B}(x, r)$ *denote a box of radius* r *centered at* $x \in M$. *Define the box collection* $\widehat{T(B)} = \{B' \in \mathcal{B} : B' \cap \bigcup_{x \in G} \mathbf{B}(Tx, r) \neq \emptyset\}$. *Then* $\{B' \in \mathcal{B} : B' \cap T(B) \neq \emptyset\} \subset \widehat{T(B)}$.

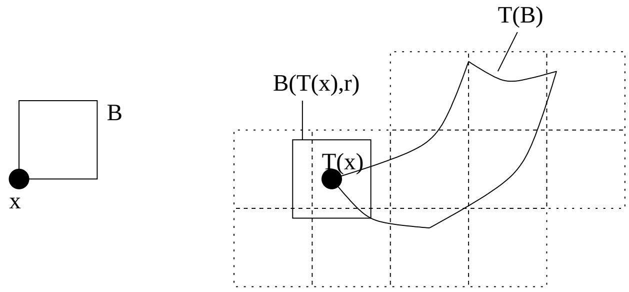

Fig. 13. Illustration of Proposition 22.

The collection $\widehat{T(B)}$ is constructed by placing a box of radius r, centered at every Tx, $x \in G$, and checking which boxes in $\mathcal{B}$ are intersected by these boxes, see also Figure 13. This check is very simple, as one only needs to include all boxes into $\widehat{T(B)}$ which contain a vertex of the boxes $B(T(x_i), r)$, $i = 1, \ldots, q$.

In GAIO this construction is realized by a special choice of the test points which have to assigned to a `Tree` object. Recall that for the non-rigorous approach one may chose

```
tree.domain_points = Points('Edges', dim, m)
tree.image_points = Points('Center', dim)
```

where `dim` denotes the dimension of phase space and m the suggested number of points. Now for the rigorous approach we instead use the commands

```
tree.domain_points = Points('Lipschitz', dim)
tree.image_points = Points('Vertices', dim)
```

An alternative, even more efficient way of constructing a grid of test points is to align the points within B along directions given by the right singular vectors of the matrix L, with the components of h directly related to the singular values. This approach reduces the required number of test points for the determination of a covering of the image of B under T. For a detailed description of the methods mentioned in this section see [18].

B Example Model Files

Each model is defined in a single C–file which is compiled into object code by a C–compiler and then transformed into a shared object by the linker of the machine. This shared object can then be loaded into GAIO.

We give two example model files to show how one produces a model in practice. There are two main situations; the first is where our dynamical system is governed by a discrete map, and the second is where our system is a flow generated by a vector field. In the former case, the map will form the "right hand side" in our model files, while in the latter case, it is the vector field.

B.1 Map

The following is the code of `henon.c`, a C–file that produces a model file for the Hénon map.

```
char *name = "My Henon map";
char *typ = "map";
int   dim  = 2;
int   paramDim = 2;
char *paramNames[] = { "a", "b" };
double a = 1.3, b = 0.2;
double c[2] = { 0, 0 };
double r[2] = { 3, 3 };
double tFinal = 1;

void rhs(double *x, double *u, double *y) {
  y[0] = 1 - a*x[0]*x[0] + x[1];
  y[1] = b*x[0];
}
```

Except for `a` and `b`, which are obviously specific to the Hénon map, these are the variables and function(s) that have to present in each model file. The file now has to be compiled into object code by the command

```
cc -c henon.c
```

and the resulting object file `henon.o` has to be converted into a shared object by issuing

```
ld -shared -o henon.so henon.o
```

Note that the linker flag `-shared` is platform dependent. Some specific choices are given in the following table:

Linux	`-shared`
Solaris	`-G`
OSF or Irix	`-shared -expect_unresolved`

Additionally one has to provide a Lipschitz estimate on the right hand side if one wants to use the `Lipschitz` test points as mentioned in section 2.2 and explained in Appendix A. The corresponding function must be called `lip` as in the following example for the Hénon map.

```
#define max(x,y) (x>y ? x : y)
void lip(double *c, double *r, double *L) {
  L[0] = 2.0*fabs(a)*max(fabs(c[0] + r[0]), fabs(c[0] - r[0]));
  L[1] = 1.0;
  L[2] = fabs(b);
  L[3] = 0.0;
}
```

Finally it is possible to additionally supply some special point in state space via the function `fixed_point`. In most cases this will be a fixed point of the underlying dynamical system. For the Hénon map we define:

```
#include <math.h>   /* defines sqrt() */

void fixed_point(double *x) {
  double t = (b-1)/(2*a);
  x[0] = t + sqrt(t*t + 1/a);
  x[1] = b*x[0];
}
```

B.2 Flow

The following is a C–file that produces a model file for the Lorenz flow, with three free parameters.

```
#include <math.h>

char *name = "The Lorenz system";
char *typ = "ode";
int   dim  = 3;
int   paramDim = 3;
char *paramNames[] = { "sigma", "rho", "beta" };
double  sigma = 10, rho = 28, beta = 2.666666666;
double c[3] = { 0, 0, 27 };
double r[3] = { 30, 30, 40 };
double tFinal = 0.2;

void rhs(double *x, double *u, double *y) {
  y[0] = sigma*(x[1]-x[0]);
  y[1] = rho*x[0] - x[1] - x[0]*x[2];
  y[2] = x[0]*x[1] - beta*x[2];
}

void fixed_point(double *x) {
  x[0] = sqrt(beta*(rho-1));
  x[1] = x[0];
  x[2] = rho-1;
}
```

C Frequently Used GAIO Commands

Here we give a short summary of more frequently used commands. A complete description is distributed together with the software.

C.1 General Commands

`model = Model('name')`

Loads the model object from the file `name.so`.

`integ = Integrator('Map')`

Loads the `Map` integrator (always used with discrete systems). For flows there are different Runge-Kutta schemes available, including `'Euler'`, `'RungeKutta4'`, `'DormandPrince853'`, `'MidpointRule'` or `'Gauss6'`.

`tree = Tree(center, radius)`

Constructs a tree object; `center` and `radius` determine the outer box Q.

`tree.depth`

Returns the current maximal depth of `tree`.

`tree.count(depth)`

Returns the number of boxes in the `tree` at the given depth.

`points = Points('InnerGrid', dim, m)`

Loads a set of `m` test points of dimension `dim`. The type `'InnerGrid'` sets the points in a uniform grid covering the box, but with all points in the interior of the box. Other possible point types are `'Grid'`, where points are also placed around the boundary of the box, `'MonteCarlo'`, which randomly distributes points (chosen from a uniform distribution) over the box and `'Edges'` which refers to test points placed on the boundary of the boxes. To use rigorous covering algorithms, one has to use the point type `'Lipschitz'` for the domain points (this is currently restricted to maps; also note that the model has to supply the function `lip` in this case).

`tree.domain_points = points`

Defines the domain test points for each box.

`tree.image_points = points`

Defines the image test points as explained in Appendix A. If one wishes to use rigorous covering algorithms, the point style `'Vertices'` has to be used for the image points.

`tree.boxes(depth)`

Returns a $(2d+2) \times n$ matrix representing the box collection on the given `depth` of the `tree`, where n is the number of boxes on this depth and d is the dimension of state space (as defined by the model). Each column of this matrix corresponds to one box, where the first d rows specify the center, the second d the radius of the box, the (2d+1)st row its flags and the last its color.

C.2 Commands for Invariant Sets and Global Attractors

`rga(tree [, steps])`

Performs `steps` (default = 1) steps of the subdivision algorithm for the computation of the relative global attractor (Algorithm 1).

`crs(tree [, steps])`

Performs several steps of the subdivision algorithm for the computation of the chain recurrent set (Algorithm 2).

C.3 Commands for Invariant Manifolds

`tree.insert(x, depth)`

Inserts the box containing the point `x` at the given depth into the tree.

`gum(tree, depth [, steps])`

Performs several steps of the continuation algorithm for the computation of global unstable manifolds (Algorithm 3) at the given depth.

C.4 Commands for Transfer Operators, Invariant measures and Almost Invariant Sets

`P = tree.matrix(method, depth)`

Computes the transition matrix as a finite-dimensional approximation of the Perron-Frobenius operator on the given depth of the tree using

1. test points, if `method` is a points object;
2. the exhaustion method, if `method='exhaustion'`.

`[v, lambda] = eigs(P, n)`

Finds the `n` eigenvectors `v` and eigenvalues `lambda` of `P` with the largest modulus. See the documentation to `eigs` for further options and details.

References

1. M. Dellnitz, G. Froyland, and S. Sertl. On the isolated spectrum of the Perron-Frobenius operator. *Nonlinearity*, 13(4):1171–1188, 2000.
2. M. Dellnitz, M. Golubitsky, and M. Nicol. *Symmetry of attractors and the Karhunen-Loéve decomposition*, pages 73–108. Number 100 in Applied Mathematical Sciences. Springer-Verlag, 1994.
3. M. Dellnitz and A. Hohmann. The computation of unstable manifolds using subdivision and continuation. In H.W. Broer, S.A. van Gils, I. Hoveijn, and F. Takens, editors, *Nonlinear Dynamical Systems and Chaos*, pages 449–459. Birkhäuser, *PNLDE* 19, 1996.
4. M. Dellnitz and A. Hohmann. A subdivision algorithm for the computation of unstable manifolds and global attractors. *Numerische Mathematik*, 75:293–317, 1997.
5. M. Dellnitz, A. Hohmann, O. Junge, and M. Rumpf. Exploring invariant sets and invariant measures. CHAOS: An Interdisciplinary Journal of Nonlinear Science, 7(2):221, 1997.
6. M. Dellnitz and O. Junge. On the approximation of complicated dynamical behavior. *SIAM J. Numer. Anal.*, 36(2):491–515, 1999.

7. M. Dellnitz, O. Junge, M. Rumpf, and R. Strzodka. The computation of an unstable invariant set inside a cylinder containing a knotted flow. In *Proceedings of Equadiff '99, Berlin*, 2000.
8. P. Deuflhard, M. Dellnitz, O. Junge, and Ch. Schütte. *Computation of essential molecular dynamics by subdivision techniques*, pages 98–115. Number 4 in Lecture Notes in Computational Science and Engineering. Springer-Verlag, 1998.
9. J. Ding and A. Zhou. Finite approximations of Frobenius-Perron operators. A solution of Ulam's conjecture to multi-dimensional transformations. *Physica D*, 92(1–2):61–68, 1996.
10. R.W. Easton. *Geometric Methods for Discrete Dynamical Systems.* Number 50 in Oxford engineering science. Oxford University Press, New York, 1998.
11. M. Eidenschink. *Exploring Global Dynamics: A Numerical Algorithm Based on the Conley Index Theory.* PhD thesis, Georgia Institute of Technology, 1995.
12. Euclid. *Elements.* Book X, (first Proposition).
13. G. Froyland. Finite approximation of Sinai-Bowen-Ruelle measures of Anosov systems in two dimensions. *Random & Computational Dynamics*, 3(4):251–264, 1995.
14. G. Froyland. Approximating physical invariant measures of mixing dynamical systems in higher dimensions. *Nonlinear Analysis, Theory, Methods, & Applications*, 32(7):831–860, 1998.
15. G. Froyland and M. Dellnitz. Detecting and locating near-optimal almost-invariant sets and cycles. In preparation.
16. R. Guder, M. Dellnitz, and E. Kreuzer. An adaptive method for the approximation of the generalized cell mapping. *Chaos, Solitons and Fractals*, 8(4):525–534, 1997.
17. R. Guder and E. Kreuzer. Control of an adaptive refinement technique of generalized cell mapping by system dynamics. *J. Nonl. Dyn.*, 20(1):21–32, 1999.
18. O. Junge. *Mengenorientierte Methoden zur numerischen Analyse dynamischer Systeme.* PhD thesis, University of Paderborn, 1999.
19. O. Junge. Rigorous discretization of subdivision techniques. In *Proceedings of Equadiff '99, Berlin*, 2000.
20. G. Keller and C. Liverani. Stability of the spectrum for transfer operators. Preprint, 1998.
21. H. Keller and G. Ochs. Numerical approximation of random attractors. In *Stochastic dynamics*, pages 93–115. Springer, 1999.
22. R.Z. Khas'minskii. Principle of averaging for parabolic and elliptic differential equations and for Markov processes with small diffusion. *Theory of Probability and its Applications*, 8(1):1–21, 1963.
23. Y. Kifer. *Random Perturbations of Dynamical Systems*, volume 16 of *Progress in Probability and Statistics.* Birkhäuser, Boston, 1988.
24. A. Lasota and M.C. Mackey. *Chaos, Fractals, and Noise. Stochastic Aspects of Dynamics*, volume 97 of *Applied Mathematical Sciences.* Springer-Verlag, New York, second edition, 1994.
25. T.-Y. Li. Finite approximation for the Frobenius-Perron operator. A solution to Ulam's conjecture. *Journal of Approximation Theory*, 17:177–186, 1976.
26. K. Mehlhorn. *Data Structures and Algorithms.* Springer, 1984.

27. G. Osipenko. Construction of attractors and filtrations. In K. Mischaikow, M. Mrozek, and P. Zgliczynski, editors, *Conley Index Theory*, pages 173–191. Banach Center Publications 47, 1999.
28. C. Robinson. *Dynamical Systems: Stability, Symbolic Dynamics, and Chaos.* CRC, Boca Raton, 1995.
29. M. Rumpf and A. Wierse. GRAPE, eine objektorientierte Visualisierungs– und Numerikplattform. *Informatik, Forschung und Entwicklung*, 7:145–151, 1992.
30. Ch. Schütte. *Conformational Dynamics: Modelling, Theory, Algorithm, and Application to Biomolecules.* Habilitation thesis, Freie Universität Berlin, 1999.
31. E.C. Zeeman. Stability of dynamical systems. *Nonlinearity*, 1:115–155, 1988.