GAIO.jl

Preparing for a 1.0 release using julia 1.9

April Herwig

Department of Mathematics
Technical University of Munich

Dynamical systems

Consider a continuous map

$$f : \mathbb{R}^d \to \mathbb{R}^d.$$

The map defines a discrete dynamical system by iteration:

$$x_{k+1} = f(x_k), \qquad k = 0, 1, 2, \dots$$

Basic question: *What is the fate of some x_0 as $k \to \infty$?*

Motivation

Attractors

Definition 1. *An invariant set A is attracting if there is a neighborhood U of A such that for every open set $V \supset A$ there is $K \in \mathbb{N}$ such that*

$$f^k(U) \subset V \quad \text{for all } k \geq K.$$

Proposition 1. *If A is a closed attracting set then*

$$A = \bigcap_{k \in \mathbb{N}} f^k(U).$$

Basic idea: Successively refine an approximation of A using subdivision

Computing Attractors
The subdivision algorithm

Generate a sequence $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \ldots$ of finite families of compact sets as follows:

Let $\mathcal{B}_0 = \{Q\}$, $\theta \in (0,1)$. For $k = 1, 2, \ldots$ do

- construct $\hat{\mathcal{B}}_k$ such that

$$|\hat{\mathcal{B}}_k| = |\mathcal{B}_{k-1}| \quad \text{and} \quad \operatorname{diam} \hat{\mathcal{B}}_k \leq \theta \cdot \operatorname{diam} \mathcal{B}_{k-1}.$$

- set

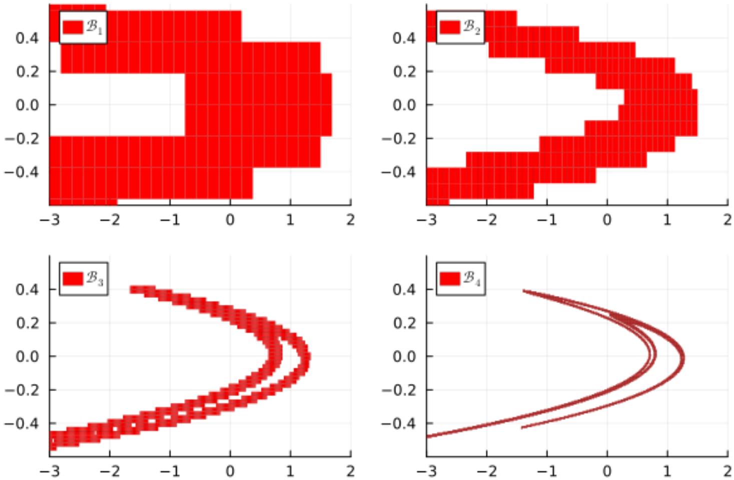
$$\mathcal{B}_k = f(\hat{\mathcal{B}}_k) \cap \hat{\mathcal{B}}_k, \quad \text{where}$$

$$f(\hat{\mathcal{B}}_k) \cap \hat{\mathcal{B}}_k \stackrel{def}{=} \{B \in \hat{\mathcal{B}}_k \mid \exists B' \in \hat{\mathcal{B}}_k : f(B') \cap B \neq \varnothing\}.$$

Theorem 1. *$|\mathcal{B}_k| \to A_Q$ as $k \to \infty$ in the Hausdorff metric.*

Computing Attractors

The subdivision algorithm



Representation of Cubical Sets

Simple idea: partition the domain into equally sized grid of hypercubes

```
struct Box{N,T}
  lowbound::SVector{N,T}
  highbound::SVector{N,T}
end

struct BoxPartition{B <: Box}
  domain::B
  size::CartesianIndex{N}
end

struct BoxSet{P <: BoxPartition, S <: AbstractSet{<:CartesianIndex}}
  partition::P
  cartesian_indices::S
end
```

Representation of Cubical Complexes

We can use the built-in set data types and setwise operations for `BoxSets` using multiple-dispatch

```
function Base.⊆(B₁::BoxSet, B₂::BoxSet)
  B₁.partition == B₂.partition &&
    B₁.cartesian_indices ⊆ B₂.cartesian_indices
end

function Base.rand(B::Box{N,T}) where {N,T}
  l = B.lowbound;  h = B.highbound
  l + (h-l) .* rand(SVector{N,T})
end
```

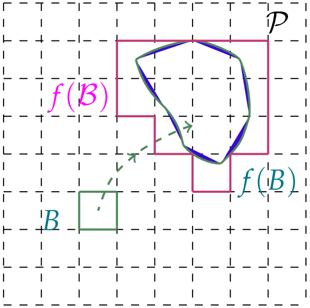
Combined with `BoxSet`, the `BoxPartition` serves as a σ-algebra over the domain. We can generate sets with `cover`:

```
cover(partition, points), cover(partition, other_boxset)
```

Cell Mapping

So how do we compute

$$f(\mathcal{B}) = \{B \in \mathcal{P} \mid \exists B' \in \mathcal{B} : f(B') \cap B \neq \varnothing\} \ ?$$



Cell Mapping
test point sampling

```
function map_boxes(f, source::BoxSet{}; n_samples)

  B() = empty(source)                     # Initialize empty BoxSet
  P = source.partition

  @floop for box in source
    for k = 1:n_samples
      p = rand(box)                       # Generate sample point
      image_p = f(p)
      hit = cover(P, image_p)             # Box in P covering the point fp
      @reduce(image = B() ∪ hit)          # Each thread collects hits,
    end                                   # after loop completion the
  end                                     # result is reduced by ∪

  return image
end
```

Cell Mapping

test point sampling

- predefine test points and (affine) transform them to $B \in \mathcal{B}$ as needed
- memory-efficient "lazy" test point sampling with `Generators`
- ensure type-stability with `FunctionWrappers` to shorten generated code
- spread load across multiple compute threads using `@floop` macro
- collect hits and reduce per-thread result into single result using `@reduce` macro
- "embarrasingly parallel" - can harness the GPU using CUDA.jl

Usage: Constructing an Attractor algorithm

Compare the pseudocode algorithm

Require: $f,\ \mathcal{B}_0,\ n_{\text{steps}}$

1: **for** $k = \{1, \ldots, n_{\text{steps}}\}$ **do**

2: $\quad \mathcal{B}_k \leftarrow \textsc{subdivide}(\mathcal{B}_{k-1})$

3: $\quad \mathcal{B}_k \leftarrow \mathcal{B}_k \cap f(\mathcal{B}_k)$

4: **return** $\mathcal{B}_n$

to the julia implementation

```julia
function relative_attractor(f::BoxMap, 𝓑::BoxSet, steps)
  for k = 1:steps
    𝓑 = subdivide(𝓑)
    𝓑 = 𝓑 ∩ f(𝓑)
  end
  return 𝓑
end
```