The Journal of Open Source Software

GAIO.jl - A concise Julia package for the global analysis of dynamical systems

April Herwig [1], **Oliver Junge** [1], **and Michael Dellnitz**[2]

1 Technical University of Munich, Germany **2** University of Paderborn, Germany

DOI: 10.21105/joss.09266

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @dawbarton
- @tomkimpson

Submitted: 18 September 2025
Published: 12 December 2025

Summary

We provide an implementation of set-oriented numerical methods (Dellnitz et al., 2001; Dellnitz & Junge, 2002) in a Julia package. The package enables the rigorous computation of invariant sets (e.g., chain recurrent sets, attractors, invariant manifolds) in dynamical systems and provides discretizations of the transfer and the Koopman operator, enabling the computation of, e.g., invariant measures, almost invariant, cyclic and coherent sets. We showcase the features of GAIO.jl by some classical computations of (almost) invariant sets and invariant measures in dynamical systems.

Statement of Need

The original implementation (Junge, 2015) in C/Python/MATLAB suffered from (1) a complex syntax which required unnecessary technical knowledge on the underlying data structures and (2) the two language problem, rendering extensions and maintenance difficult. The data structures and the algorithmic interface have been completely redesigned for GAIO.jl. As a result, the code is concise and close to the mathematical formulation of the underlying algorithms. At the same time, the performance is equal or better.

Global Analysis of Dynamical Systems

A discrete-time dynamical system is given by a map $f : X \to X$ on some domain X. For simplicity, we here assume a compact $X \subset \mathbb{R}^d$ and f to be a homeomorphism.

Geometric/Topological analysis

A set $S \subset X$ is *forward invariant* if $f(S) \subset S$, *backward invariant* if $f^{-1}(S) \subset S$ [1], and *invariant* if it is both forward and backward invariant.

The *maximal invariant set* contained in some set S is

$$\mathrm{Inv}(S) = \{x \in S \mid f^k(x) \in S \text{ for all } k \in \mathbb{Z}\}.$$

It follows from the definition that $\mathrm{Inv}(S)$ contains all other invariant sets which are contained in S. If $S \subset X$ is forward invariant, then $\mathrm{Inv}(S) = \bigcap_{k\geq 0} f^k(S)$. If S is not forward invariant, the set $A_S := \bigcap_{k\geq 0} f^k(S)$ is called the *attractor relative to* S (Dellnitz & Hohmann, 1997). This characterization leads to a natural algorithm for approximating A_S by repeatedly tightening a cover by finite collections of subsets of X (Dellnitz & Hohmann, 1997):

[1] $f^{-1}(S) = \{x \in X : f(x) \in S\}$ denotes the *preimage* of S.

Algorithm 1: Given a partition $\mathcal{X}$ of X into (essentially) disjoint sets and some covering $\mathcal{A} \subset \mathcal{X}$ of A_S, repeat the following two steps until a prescribed diameter of the partition elements is reached:

1. Refine $\mathcal{X}$ into a strictly finer partition $\mathcal{X}'$ (i.e., such that $\operatorname{diam}(\mathcal{X}') \leq \theta \cdot \operatorname{diam}(\mathcal{X})$ for some fixed $\theta < 1$). Let $\mathcal{A}'$ be the corresponding refinement of the covering $\mathcal{A}$ of A_S.
2. Map the refined covering forward under f, i.e., cover [2] $f(|\mathcal{A}'|)$ by elements of $\mathcal{X}'$. Intersect this covering with $\mathcal{A}'$.

In GAIO.jl, partitions resp. coverings are implemented as collections of *boxes*, i.e., multidimensional intervals of the form $[\ell_1, u_1] \times \ldots \times [\ell_d, u_d]$.

Example: A four wing attractor

We now implement the algorithm to compute a covering of the attractor of the system proposed in (Wang et al., 2009). Let f be the time-2 flow map of the system, discretized using 20 steps of the standard Runge-Kutta fourth order method:

```
using GAIO
const a, b, d = 0.2, -0.01, -0.4
v((x,y,z)) = @. (a*x+y*z, d*y+b*x-z*y, -z-x*y)
f(x) = rk4_flow_map(v, x, 0.01, 20)
```

For demonstration we compute a long trajectory, shown in the left plot in Figure 1.

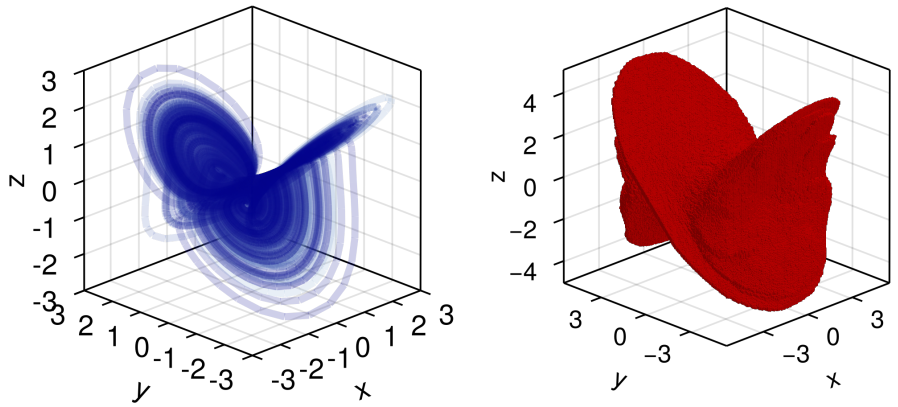

Figure 1: A trajectory of the four wing system and its (relative) attractor.

For step 1. in Algorithm 1, we construct a `BoxGrid` `PX` which partitions the domain $X = [-5, 5]^3$ into a $2 \times 2 \times 2$ grid of boxes. One can think of `PX` as the power set[3] $\mathcal{P}(\mathcal{X})$ of the partition $\mathcal{X}$:

```
center, radius = (0.,0.,0.), (5.,5.,5.)
X = Box(center, radius)
PX = BoxGrid(X, (2, 2, 2))
```

We next construct a (coarse) initial covering, the `BoxSet` `A` consisting of all boxes in `PX`:

```
A = cover(PX, :)
```

The command

```
subdivide(A, k)
```

[2] $|\mathcal{A}| = \bigcup_{\mathcal{X} \in \mathcal{A}} \mathcal{X}$

[3] The *power set* $\mathcal{P}(S)$ of some set S is the set of all subsets of S.

can then be used for refining the `BoxSet A` by bisecting each box in the k-th coordinate direction.

To implement step 2. in Algorithm 1, note that the flow map f induces a map $F : \mathcal{P}(\mathcal{X}) \to \mathcal{P}(\mathcal{X})$ via

$$F(\{\chi\}) = \{\hat{\chi} \in \mathcal{X} \mid \hat{\chi} \cap f(\chi) \neq \emptyset\}.$$

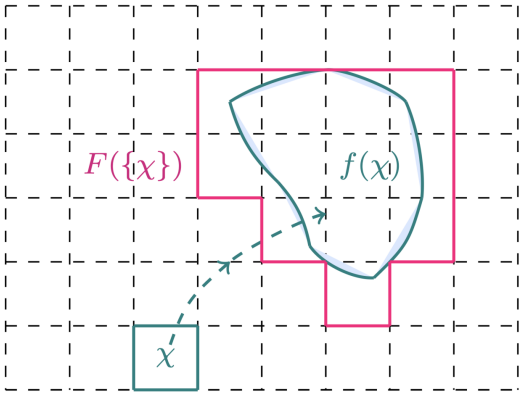

Figure 2: Lifting point maps to maps on boxes.

GAIO.jl provides multiple methods for approximating F (Junge, 2000). An intuitive method is to randomly sample some points within a box, and map each point with f. Such an approximation can be instantiated by

```
F = BoxMap(:montecarlo, f, PX)
```

We can now fully implement the algorithm:

```
function relative_attractor(F, A; steps)
    for k in 1:steps
        A = subdivide(A, k%3+1)
        A = F(A) ∩ A
    end
    return A
end
```

We cycle through the coordinate direction in which we bisect boxes in each step. We invoke `relative_attractor` on the initial covering `A`. The result is shown in the right plot in Fig. 1:

```
A = relative_attractor(F, A, steps=21)
plot(A)
```

Other algorithms in GAIO.jl include ones for, e.g., computing forward, backward and maximal invariant sets, (un-)stable manifolds, chain-recurrent sets and Morse decompositions.

Statistical analysis

$f : X \to X$ induces a map $f_\sharp : \mathcal{M} \to \mathcal{M}$ on measures[4] via

$$f_\sharp \mu := \mu \circ f^{-1}.$$

This is a linear Markov operator, the *transfer operator*. Much information about macroscopic features of the dynamics of f can be extracted from eigenmeasures of $f_\sharp$ at eigenvalues with modulus close to one (Dellnitz & Junge, 1998).

[4] $\mathcal{M}$ denotes the space of finite, complex valued Borel measures on X.

One can approximate $\mu \in \mathcal{M}$ by a discrete measure

$$\mu_g(S) = \sum_{j=1}^{n} g_j \frac{m(\chi_j \cap S)}{m(\chi_j)},$$

where $\{\chi_1, \chi_2, \dots, \chi_n\}$ enumerates the partition $\mathcal{X}$, $g = (g_1, \dots, g_n) \in \mathbb{C}^n$ and m is Lebesgue measure on $\mathbb{R}^d$. The coefficients g of an approximate invariant measure μ_g should then satisfy

$$g_i = \mu_g(\chi_i) \overset{!}{=} f_\sharp \mu_g(\chi_i) = \sum_{j=1}^{n} g_j \underbrace{\frac{m(\chi_j \cap f^{-1}(\chi_i))}{m(\chi_j)}}_{=:(F_\sharp)_{ij}}.$$

$F_\sharp \in \mathbb{R}^{n \times n}$ defines a Markov chain on $\mathcal{X}$ and is our approximation of $f_\sharp$ on $\mathcal{M}_n = \{\mu_g : g \in \mathbb{C}^n\}$. It can be computed by approximating the transition probabilities $(F_\sharp)_{ij}$ e.g. using random sample points.

We compute $F_\sharp$ on the covering constructed above and then compute part of its spectrum:

```
F_sharp = TransferOperator(F, A, A)
lambda, ev, n_converged = eigs(F_sharp)
```

The eigenvalue 1 is simple, and the corresponding eigenvector approximates an invariant measure shown in the left plot in Fig. 3. Such a (natural) invariant measure (Young, 2002) quantifies the statistics of typical trajectories: Regions of phase space which are visited more often by such trajectories receive more μ-mass. In the right of Fig. 3, we show the eigenmeasure at the second largest real eigenvalue $\lambda \approx 0.978$. Its sign structure decomposes the attractor into two almost invariant sets (Dellnitz & Junge, 1998), i.e., two sets A_-, A_+ for which the invariance ratio $m(A_+ \cap f^{-1}(A_+))/m(A_+)$ (resp. A_-) is close to 1.

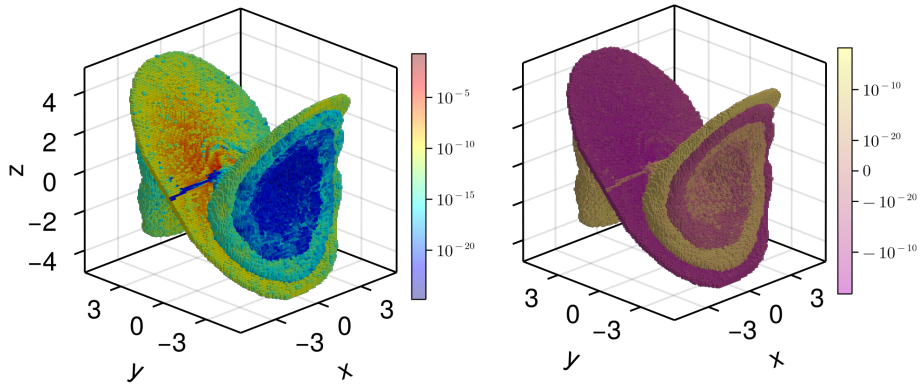

Figure 3: Natural invariant measure of f and eigenmeasure for the eigenvalue 0.978

GAIO in the Julia Language

GAIO was originally developed in the 90s in C (Dellnitz et al., 2001). Interfaces were written in Numerical Python (Harris & al., 2020) and MATLAB, and 3D plotting was done by the dedicated software GRAPE (Rumpf & Wierse, 1992). This architecture (Fig. 4) was hard to maintain. GAIO was a picture book incarnation of the *two language problem* (Bezanson et al., 2017).

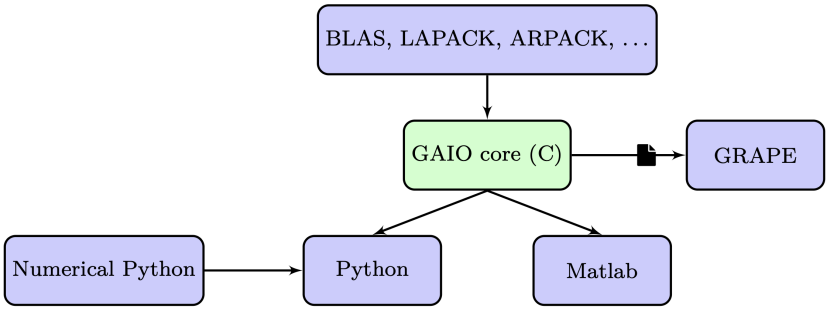

Figure 4: An earlier software architecture.

GAIO was fully redesigned in the Julia language starting in 2020. The reasons were:

- *Solving the two language problem.* The set-oriented techniques require many evaluations of the map f. Compilation into fast machine code is necessary if one wishes to apply the methods to a non-trivial problem.
- *Abstraction of the syntax.* The original GAIO code required the user to be aware of the internal data structures. Box collections were stored in a binary tree, flags in its nodes were used in order to implement the box map $F : \mathcal{P}(\mathcal{X}) \to \mathcal{P}(\mathcal{X})$. The original GAIO/MATLAB implementation of Algorithm 1 reads:

```
function relative_attractor(tree, f, steps)
    for i = 1:steps,
        tree.set_flags('all', to_be_subdivided);
        tree.subdivide(to_be_subdivided);
        b = tree.boxes(-1);
        while (~isempty(b))
            c = b(1:d);
            r = b(d+1:2*d);
            P = X*diag(r)+ones(size(X))*diag(c);
            tree.set_flags(f(P)', hit);
            b = tree.next_box(-1);
        end
        tree.remove(hit);
    end
```

In contrast, in GAIO.jl, details on how box collections are stored and how the box map is realized are essentially hidden from the user. As a result, the implementation of the algorithms is very close to its mathematical formulation.

Fitting into Julia's ecosystem

Another reason to use Julia for GAIO was the large package ecosystem. A particular example is CUDA.jl: The sample-point method for mapping cells is *embarrassingly parallel* (Herlihy et al., 2020). It has therefore been a long-standing desire to utilize the GPU to perform these computations. Under the previous architecture this would have to be written in CUDA's native C interface. With CUDA.jl, it can be written as a generic kernel whose length is the same as the standard code. The algorithms in GAIO.jl receive up to a 200-fold (Herwig, 2022) performance boost without ever sacrificing readability.

Conclusion

GAIO.jl is introduced via an example of a three dimensional dynamical system. The package has been redesigned to balance high performance and elegance, no longer needing to rely on

two languages to do so. Future work is planned to use these structures e.g., for homology computation in cubical complexes, as well as to even more tightly integrate into the existing scientific computing ecosystem.

References

Bezanson, J., Edelman, A., Karpinski, St., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Dellnitz, M., Froyland, G., & Junge, O. (2001). The algorithms behind GAIO – set oriented numerical methods for dynamical systems. In *Ergodic theory, analysis, and efficient simulation of dynamical systems* (pp. 145–174, 805–807). Berlin: Springer. https://doi.org/10.1007/978-3-642-56589-2_7

Dellnitz, M., & Hohmann, A. (1997). A subdivision algorithm for the computation of unstable manifolds and global attractors. *Numer. Math.*, *75*(3), 293–317. https://doi.org/10.1007/s002110050240

Dellnitz, M., & Junge, O. (1998). On the approximation of complicated dynamical behavior. *SIAM J. Numer. Anal.*, *36*(2), 491–515. https://doi.org/10.1137/S0036142996313002

Dellnitz, M., & Junge, O. (2002). Set oriented numerical methods for dynamical systems. In *Handbook of dynamical systems. Volume 2* (pp. 221–264). Amsterdam: Elsevier. https://doi.org/10.1016/s1874-575x(02)80026-1

Harris, C. R., & al., et. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Herlihy, M., Shavit, N., Luchangco, V., & Spear, M. (2020). *The art of multiprocessor programming*. Elsevier. https://doi.org/10.1016/C2011-0-06993-4

Herwig, A. (2022). *GAIO.jl: Set-oriented methods for approximating invariant objects, and their implementation in Julia* [B.Sc. Thesis, Technical University of Munich]. github.com/April-Hannah-Lena/schoolwork/blob/main/Bach%20Thesis/main.pdf

Junge, O. (2000). Rigorous discretization of subdivision techniques. In *Proceedings of Equadiff '99, Berlin, Germany, Vol. 2* (pp. 916–918). Singapore: World Scientific. https://doi.org/10.1142/4469

Junge, O. (2015). GAIO (Global Analysis of Invariant Objects). In *GitHub repository*. https://github.com/gaioguy/GAIO; GitHub.

Rumpf, M., & Wierse, A. (1992). GRAPE, eine objektorientierte Visualisierungs– und Numerikplattform. *Informatik – Forschung Und Entwicklung*, *7*(3), 145–151.

Wang, Z., Sun, Y., Wyk, B. J. van, Qi, G., & Wyk, M. A. van. (2009). A 3-d four-wing attractor and its analysis. *Brazilian Journal of Physics*, *39*(3), 547–553. https://doi.org/10.1590/S0103-97332009000500007

Young, L.-S. (2002). What are SRB measures, and which dynamical systems have them? *J. Stat. Phys.*, *108*(5-6), 733–754. https://doi.org/10.1023/A:1019762724717