The Journal of Open Source Software

DynamicalSystems.jl: A Julia software library for chaos and nonlinear dynamics

George Datseris[1]

1 Max Planck Institute for Dynamics and Self-Organization, Göttingen, Germany

DOI: 10.21105/joss.00598

Software

- Review
- Repository
- Archive

Submitted: 24 February 2018
Published: 14 March 2018

Introduction

Chaotic systems are everywhere (Strogatz 1995), from celestial mechanics to biology to electron transport. Not only do they cover many scales, but the phenomena that fall under the scope of "nonlinear dynamics" are multi-faceted (Faust et al. 2015). This vast extend of chaotic systems requires the use of methods from nonlinear dynamics and chaos theory in many diverse areas of science.

On the other hand, chaotic systems are not analytically solvable, therefore studying them often relies on numerical methods. Many such methods have been devised to study the many facets of nonlinear systems, but unfortunately no up-to-date and comprehensive library collecting these methods exists.

Enter DynamicalSystems.jl

DynamicalSystems.jl was created to fill this role. It is a Julia (Bezanson et al. 2017) library that offers functionality useful in the study of chaos, nonlinear dynamics and time-series analysis. **DynamicalSystems.jl** itself is also a part of the JuliaDynamics organization, similarly with the package DynamicalBilliards.jl (Datseris 2017).

The official documentation is hosted here.

DynamicalSystems.jl Goals

Our goals with this library can be summarized as:

1. Be concise, intuitive, and general.
2. Be accurate, reliable and performant.
3. Be transparent with respect to what is happening "under the hood", i.e. be clear about exactly what each function call does. We take care of this aspect in many ways; by being well-documented, giving references to scientific papers and having clear source code.

Features

- General & flexible dynamical system definition interface.
- Automatic computation of Jacobians through automatic differentiation.

- Dedicated interface for datasets, including IO.
- Delay coordinates embedding.
- Poincaré surface of sections.
- Orbit diagrams (also called bifurcation diagrams).
- Automated production of orbit diagrams for continuous systems.
- Maximum Lyapunov exponent.
- Spectrum of Lyapunov exponents.
- Generalized entropies & permutation entropy.
- Generalized dimensions and automated procedure of deducing them.
- Neighborhood estimation of points in a dataset.
- Numerical (maximum) Lyapunov exponent of a timeseries.
- Finding fixed points of a map of any order.
- Detecting and distinguishing chaos using the GALI method.

We advise the reader to visit the latest overview because new methods are constantly enriching **DynamicalSystems.jl**.

Similar existing software

We would now like to mention three other software packages that offer similar functionality to ours. We are only considering open-sourced packages in this section.

The first, TSTOOL, is aimed at nonlinear time series analysis and is implemented in MATLAB (which is Proprietary software) with a partial backend of C++. Features of TSTOOLS that are not currently offered by **DynamicalSystems.jl** are surrogate time-series and estimating suitable dimensions for delay coordinates embedding. TSTOOL operates on datasets, and thus any dataset can easily be loaded through the provided interface, but there is no definition of equations of motion. This has the result that all methods contained cannot take advantage of known equations of motion.

The second, E&F chaos (Diks et al. 2008), is implemented in LUA with a partial C/Pascal backend and is aimed at nonlinear dynamics in economics and finance. Features of E&F chaos that we do not offer are basin boundary plots, cobwebs and parameter basins. E&F chaos is the only software mentioned here that allows definition of new systems through equations of motion.

Finally, LP-VIcode (Carpintero, Maffione, and Darriba 2014) is a suite devoted solely for computing variational indicators of chaos and is written in FORTRAN77. **DynamicalSystems.jl** offers only the latest indicator from all the ones available in LP-VIcode, namely GALI (Skokos, Bountis, and Antonopoulos 2007). In addition, LP-VIcode places the severe constrain that all systems must not only be Hamiltonian, but must also have parabolic kinetic energy term, making it unusable for any other system type, Hamiltonian or not.

DynamicalSystems.jl advantages vs other packages

- It is written in purely in Julia (Bezanson et al. 2017).
  - Julia is (currently) the only open sourced & dynamic language that has performance equal to C/Fortran, allowing interactivity without adding computational costs.
- Fully exposes the differential equation solvers of continuous systems, and gives the user the (possible) full control over them through the DifferentialEquations.jl suite (Rackauckas and Nie 2017).
- Offers the widest range of methods.

- Transparent and small source code.
- It is concise, intuitive and general: all functions work just as well with any defined dynamical system.
- Extendable; adding new systems or algorithms requires minimal effort.
- Actively maintained and constantly growing.
- Hosted on GitHub, making interaction of users and developers easy and straightforward.

Examples

In the following examples we want to demonstrate some of the capabilities of **DynamicalSystems.jl**. In the first example will show how one can find the Lyapunov spectrum of a continuous system, while the second will show how to use delay coordinates embedding to calculate the attractor dimension from a time series. Both examples are benchmarked on a laptop with Intel(R) Core(TM) i5-6200U CPU @ 2.30 Ghz, 8GB RAM, 64-bit Windows 10 operating system and Julia version v0.6.2.

Lyapunov spectrum of a continuous system

The first step is to create a `DynamicalSystem` structure:

```
# Pkg.add("DynamicalSystems")
using DynamicalSystems
# Define Lorenz system: equations take the current state `u`
# with parameters `p` at time `t` and return an SVector
# with the derivatives.
@inline @inbounds function lorenz(u, p, t)
      = p[1];   = p[2];   = p[3]
    du1 =  *(u[2]-u[1])
    du2 = u[1]*( -u[3]) - u[2]
    du3 = u[1]*u[2] -  *u[3]
    return SVector{3}(du1, du2, du3)
end
# A function for the Jacobian is useful but not necessary;
# If it is not given, automatic differentiation is used
@inline @inbounds function lorenz_jac(u, p, t)
     ,  ,   = p
    J = @SMatrix [-     0;
      - u[3]  (-1)  (-u[1]);
    u[2]   u[1]  - ]
    return J
end
# typical chaotic initial condition:
u0=[0.0, 10.0, 0.0]
# parameters with strange attractor:
p = [10, 28, 8/3]

# Create the dynamical system structure
ds = ContinuousDynamicalSystem(lorenz, u0, p, lorenz_jac)

3-dimensional continuous dynamical system
 state:     [0.0, 10.0, 0.0]
```

```
 e.o.m.:    lorenz
 in-place?  false
 jacobian:  lorenz_jac
```

This structure can now be given to functions like `lyapunovs`

```
# Calculate the full lyapunov spectrum by doing QR-decomposition 2000 times
# and evolving in between for 2.0 units of time
s = lyapunovs(ds, 2000; dt = 2.0)
```

```
Float64[3]
0.904…
-0.000115…
-14.6…
```

```
# benchmark:
using BenchmarkTools
@btime lyapunovs($ds, 2000; dt = 2.0);
```

```
  225.790 ms (348 allocations: 35.58 KiB)
```

Of course specific numbers change from run to run (random initialized Q-matrix).

It is worth noting that the default differential equation solver used for continuous systems is a 9th order Vernier algorithm, with absolute and relative tolerances of `1e-9`. This must be taken into account before comparing benchmarks with other software. However, because of the excellent interaction of our library with the DifferentialEquations.jl suite (Rackauckas and Nie 2017), it is very straight forward to use a different ODE solver.

Information Dimension from Delay Coordinates Embedding

Here we show how to analyze timeseries with **DynamicalSystems.jl**. We first generate example timeseries of the Hénon map (Henon 1976) and then calculate the fractal dimension of the underlying attractor.

```
ds = Systems.henon() # load one of the predefined systems

# Get a trajectory of the system:
traj = trajectory(ds, 100000)

# A timeseries is univariate:
ts = traj[:, 1] # first column of dataset

# Now perform delay coordinates embedding of dimension 2 and delay 2:
R = Reconstruction(ts, 2, 2)

# Now e.g. calculate the Information dimension
id = information_dim(R)

# For reference, we can compute the information dimension of the
# Henon attractor directly, because we have a trajectory
id_direct = information_dim(traj, sizes)
```

```
println("Dimensions: $(round(id, 4)), $(round(id_direct, 4))")

# Benchmark:
@btime Reconstruction($ts, 2, 2);
@btime information_dim($traj);

Dimensions: 1.1979, 1.2
  653.181  s (3 allocations: 1.53 MiB)
  590.347 ms (1819827 allocations: 118.57 MiB)
```

We note that the function `information_dim`, and other similar ones, computes a lot of automated steps by measuring entropies at many different partition sizes (by default 12) and deducing a scaling slope. All of these parameters can be changed by the user.

Acknowledgements

We would like to thank Chris Rackauckas (Rackauckas and Nie 2017) for excellent help regarding the integration of the DifferentialEquations.jl suite to our library. We thank Takafumi Arakaki for contributing a method that computes the permutation entropy of a timeseries. We thank Sebastian Micluța-Câmpeanu for minor testing of continuous systems methods.

References

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. "Julia: A Fresh Approach to Numerical Computing." *SIAM Review* 59 (1):65–98. https://doi.org/10.1137/141000671.

Carpintero, D. D., N. Maffione, and L. Darriba. 2014. "LP-VIcode: A program to compute a suite of variational chaos indicators." *Astronomy and Computing* 5:19–27. https://doi.org/10.1016/j.ascom.2014.04.001.

Datseris, George. 2017. "DynamicalBilliards.Jl: An Easy-to-Use, Modular and Extendable Julia Package for Dynamical Billiard Systems in Two Dimensions." *The Journal of Open Source Software* 2 (19):458. https://doi.org/10.21105/joss.00458.

Diks, Cees, Cars Hommes, Valentyn Panchenko, and Roy Weide. 2008. "E&F chaos: A user friendly software package for nonlinear economic dynamics." *Computational Economics* 32 (1-2):221–44. https://doi.org/10.1007/s10614-008-9130-x.

Faust, Gunter, John Argyris, Gunter Faust, Maria Haase, and Rudolf Friedrich. 2015. *An Exploration of Dynamical Systems and Chaos*. Springer.

Henon, M. 1976. "A two-dimensional mapping with a strange attractor." *Communications in Mathematical Physics* 50 (1):69–77. https://doi.org/10.1007/BF01608556.

Rackauckas, Christopher, and Qing Nie. 2017. "DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia." *Journal of Open Research Software* 5 (May). https://doi.org/10.5334/jors.151.

Skokos, Ch., T.C. Bountis, and Ch. Antonopoulos. 2007. "Geometrical properties of local dynamics in Hamiltonian systems: The Generalized Alignment Index (GALI) method." *Physica D: Nonlinear Phenomena* 231 (1):30–54. https://doi.org/10.1016/j.physd.2007.04.004.

Strogatz, Steven H. 1995. *Nonlinear Dynamics and Chaos*. Perseus Books.