The Journal of Open Source Software

PathSim - A System Simulation Framework

Milan Rother [1]

1 Technische Universität Braunschweig, Germany

DOI: 10.21105/joss.08158

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @sea-bass
- @RemDelaporteMathurin

Submitted: 23 April 2025
Published: 13 May 2025

Summary

PathSim is a flexible, block-based, time-domain dynamical system simulation framework implemented in Python. It enables the modeling and simulation of complex interconnected systems from their signal flow graphs, similar to MathWorks Simulink (*Simulink, Simulation and Model-Based Design, Version R2025a*, 2025) using an object-oriented and decentralized architecture. This architectural choice distinguishes PathSim by distributing state and computation across individual `Block` components, promoting modularity, extensibility, and flexibility. Core components include user-defined or built-in `Block` objects encapsulating specific behaviors, `Connection` objects defining explicit data flow, and a `Simulation` object managing time evolution and coordination. Dynamic blocks possess their own numerical solver instances (`engine`) for state integration. PathSim incorporates advanced features: (forward mode) automatic differentiation for sensitivity analysis or gradient based optimization, discrete event handling for hybrid systems, automatic system- or block-level linearization, hierarchical modeling through subsystems, and a comprehensive suite of ODE solvers suitable for stiff problems. It requires only core scientific Python libraries: NumPy (Harris et al., 2020), SciPy (Virtanen et al., 2020), and Matplotlib (Hunter, 2007).

Statement of Need

Modeling and simulating dynamical systems is vital across many disciplines such as control engineering, chemical process engineering, and mixed signal engineering. Challenges are often the interplay of multiple *coupled* subsystems with their distinct dynamics and behaviors that have to be solved and synchronized concurrently to advance the full system simulation.

PathSim meets the need for an open source simulation framework in the block-diagram paradigm with minimal dependencies, which integrates seamlessly into the Python ecosystem. Traditional simulation tools often rely on centralized solvers, proprietary file formats, or compiled code, which can limit flexibility and extensibility within the Python ecosystem. PathSim's decentralized architecture offers distinct advantages: enhanced modularity (blocks are self-contained units), straight forward extensibility (new blocks integrate naturally without core modification), and greater flexibility in model composition and analysis. Additionally this opens up block level solver optimizations, integration with other simulation environments (co-simulation), and hardware in the loop (HiL) setups through encapsulation within blocks, as well as integration into machine-learning pipelines.

Specifically PathSim offers:

- **Open Source Alternative:** An open source alternative to legacy block-diagram system simulation frameworks.
- **Accessible Hybrid System Simulation:** Integrates event detection (zero-crossing, scheduled) directly into the block-diagram paradigm, simplifying the modeling of systems with both *continuous and discrete dynamics*.

- **Hierarchical Modeling:** Subsystem blocks that encapsulate their own blocks and connections to *manage model complexity*.
- **Gradient-Enabled Simulation:** Provides built-in *automatic differentiation* for sensitivity analysis and integration with gradient-based optimization or machine learning frameworks.
- **Unified Framework for Diverse Dynamics:** Offers a wide range of solvers, including implicit methods (ESDIRK, BDF/GEAR) for *stiff systems*.
- **Extensibility in Python:** Leverages the scientific Python ecosystem with minimal dependencies. Its architecture allows straightforward creation and integration of custom blocks.

Comparison to Existing Tools

MathWorks Simulink (*Simulink, Simulation and Model-Based Design, Version R2025a*, 2025) is the de facto industry standard for system modeling in the block diagram paradigm. It's a proprietary tool with significant licensing costs. Besides that, other equation based modeling frameworks such as `Modelica` (*Modelica Standard Library*, 1999) with its own model description language are also used but not as widely adopted.

Several Python tools for simulating dynamical systems have emerged over the years. Standard ODE solvers like `scipy.integrate.solve_ivp` (Virtanen et al., 2020) offer robust integration but lack a structured framework for modeling complex, interconnected systems, or handling discrete events natively. Users have to manually derive the govering system equations. The *Python Control Systems Library* (Fuller et al., 2021) is a popular package for modeling and optimizing dynamical systems, primarily from the control engineering perspective. The package `tbcontrol` (Sandrock, 2019) similarly focuses on control systems. Libraries like `SimuPy` (Margolis, 2017b, 2017a) provide a block-based modeling approach similar to PathSim, leveraging SymPy for symbolic definition and SciPy solvers for integration. Other frameworks like `Collimator` (Collimator.ai, 2022) offer graphical interfaces and JAX-based acceleration but require compilation and introduce dependencies beyond the standard scientific Python stack. `bdsim` (Corke, 2020) also provides block diagram simulation, based on Scipy solvers, with a strong focus on robotics but without event handling.

PathSim differentiates itself by offering a script-based block-diagram interface with a *decentralized architecture*, native integration of both *automatic differentiation* and *discrete event handling* into the full simulation loop, capable of handling *algebraic loops*, and a *built-in library* of independently implemented and verified ODE solvers (beyond wrapping SciPy). It is fully open under the *MIT License* with *minimal core dependencies*.

Architecture and Design

PathSim employs a decentralized, object-oriented design centered around three primary components:

1. **Blocks (`Block`):** Represent individual system components or operations. They encapsulate their parameters and, if stateful (like `Integrator`, `StateSpace` or `ODE`), manage their own internal state via a dedicated numerical integration engine (`engine`) instance. This contrasts with centralized approaches where a single solver manages all system states. Blocks define `update` methods for algebraic computations within a timestep and `step`/`solve` methods for interacting with their `engine` for state evolution. The `Subsystem` class shares the same interface as the base `Block` and is treated by the main simulation loop as such. This enables arbitrary encapsulation and nesting.
2. **Connections (`Connection`):** Define the explicit data flow pathways between block output ports and input ports, mirroring the connections in a block diagram.
3. **Simulation (`Simulation`):** Coordinates the overall simulation process. It maintains the list of blocks and connections. It assembles directed graphs of the block dependencies that is

used to evaluate the algebraic parts of the global system function. `Connection.update()` propagates output values to inputs, and `Block.update()` computes algebraic outputs based on current inputs and states. Algebraic loop blocks are stored in a separate graph and resolved iteratively in a second stage. The `Simulation` object then triggers the `step` (for explicit solvers) or `solve` (for implicit solvers) methods of the blocks' engines to advance their internal states. It also manages the event handling system.

PathSim Modeling Flow

PathSim is a script based modeling framework with no built-in graphical user interface. Therefore it makes sense to start from the block diagram of the system to be modeled. This section demonstrates PathSim's modeling and simulation flow on a classical example dynamical system, the *harmonic oscillator*.

The figure below shows the mechanical representation of the harmonic oscillator to the left and its block diagram to the right.

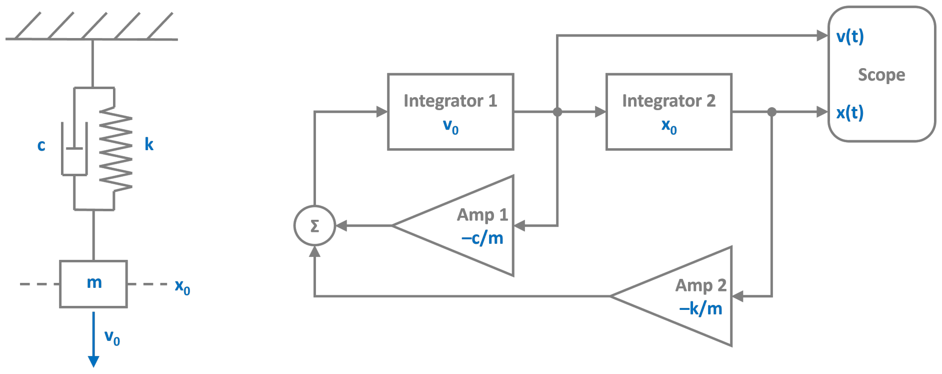

Figure 1: Mechanical representation of the harmonic oscillator to the left and its block diagram to the right as a schematic viasualization to help with the assembly of the system im PathSim (PathSim is purely script based)

Translating it to PathSim involves importing the blocks from the block library, instantiating them with their correct parameters, connecting them with the `Connection` object, and passing everything to the `Simulation`.

```python
from pathsim import Simulation, Connection
from pathsim.blocks import Integrator, Amplifier, Adder, Scope

#initial position and velocity
x0, v0 = 2, 5

#parameters (mass, damping, spring constant)
m, c, k = 0.8, 0.2, 1.5

#blocks that define the system
I1 = Integrator(v0)   # integrator for velocity
I2 = Integrator(x0)   # integrator for position
A1 = Amplifier(-c/m)
A2 = Amplifier(-k/m)
P1 = Adder()
Sc = Scope(labels=["velocity", "position"])
```

```
blocks = [I1, I2, A1, A2, P1, Sc]

#connections between the blocks
connections = [
    Connection(I1, I2, A1, Sc),
    Connection(I2, A2, Sc[1]),
    Connection(A1, P1),
    Connection(A2, P1[1]),
    Connection(P1, I1)
    ]

#create a simulation instance from the blocks and connections
Sim = Simulation(blocks, connections)

#run the simulation for some time
Sim.run(25)

#plot the results directly from the scope
Sc.plot()

#or read them out for postprocessing
time, [vel, pos] = Sc.read()
```

The Scope block enables fast plotting of the simulation results as well as their retrieval.

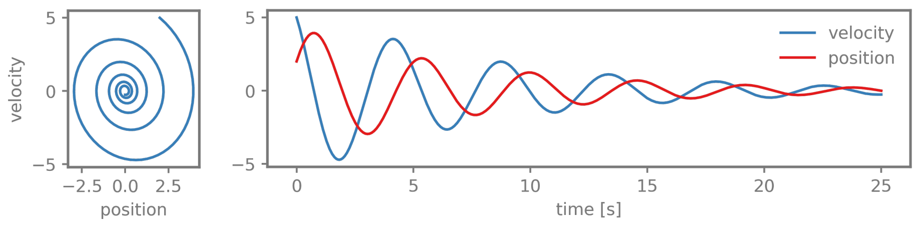

Figure 2: Simulation results plotted directly from the PathSim Scope block for the harmonic oscillator example.

There are more *examples* of dynamical system simulations present in the **PathSim** repository and, with further explanations, in the documentation. The examples cover all features **PathSim** has to offer, including hierarchical modeling through *subsystems*, examples of *stiff dynamics* with implicit solver, and discrete dynamics with *event handling*.

The figures below showcase more dynamical systems and simulation results obtained with **PathSim**.

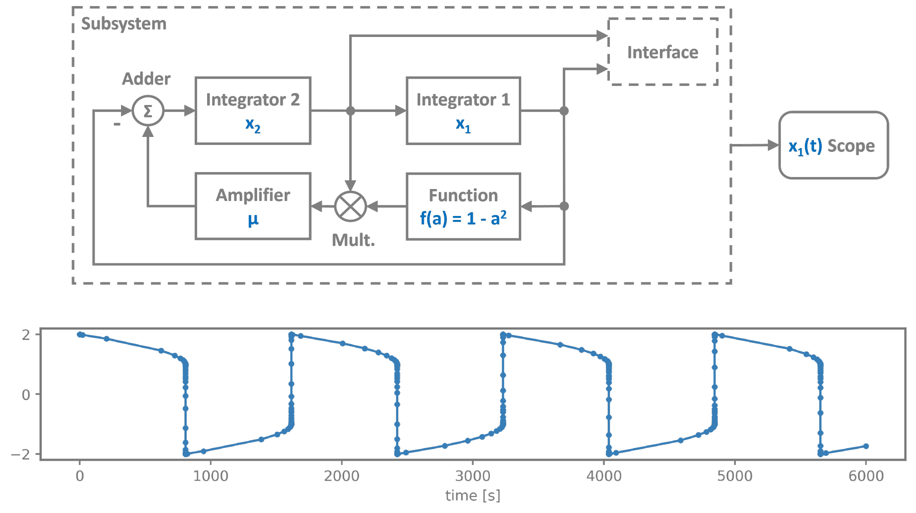

Figure 3: Visualization of the *Van der Pol* system built from distinct components as a block diagram and the simulation results for a very stiff case ($\mu = 1000$) using one of PathSim's implicit ODE solvers (`ESDIRK43`). This example showcases PathSim's ability to handle stiff systems, it is available in the repository.

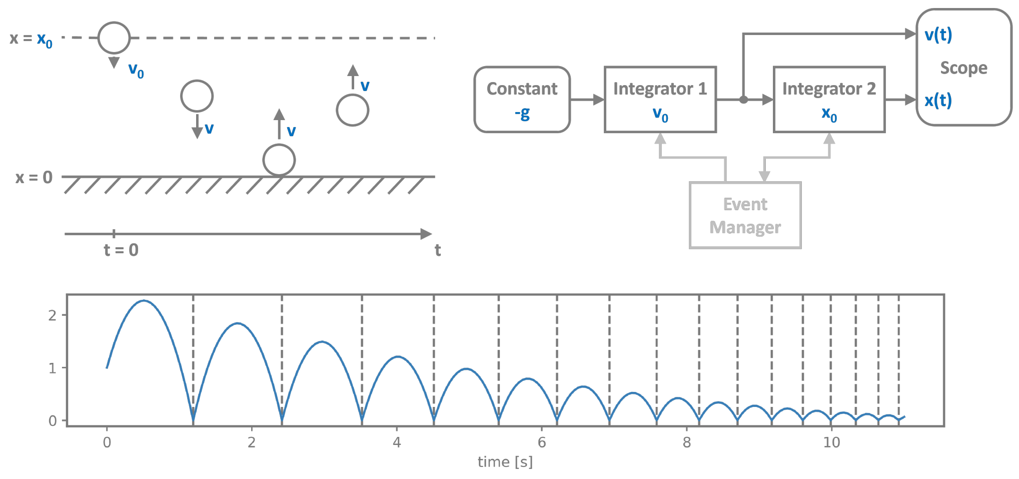

Figure 4: Visualization of the *bouncing ball*, a classical example for discrete event handling. This example showcases PathSim's event handling mechanism for detecting and resolving discrete events (*zero-crossings* in this case). It is available in the repository.

References

Collimator.ai. (2022). *Collimator.ai core simulation engine and API client*. https://pypi.org/project/pycollimator/.

Corke, P. (2020). *Bdsim: Simulate dynamic systems expressed in block diagram form using Python*. https://github.com/petercorke/bdsim.

Fuller, S., Greiner, B., Moore, J., Murray, R., Paassen, R. van, & Yorke, R. (2021). The Python Control Systems Library (python-control). *2021 60th IEEE Conference on Decision and Control (CDC)*, 4875–4881. https://doi.org/10.1109/CDC45484.2021.9683368

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Margolis, B. W. I. (2017a). *SimuPy: A framework for modeling and simulating dynamical systems*. https://github.com/simupy/simupy.

Margolis, B. W. I. (2017b). SimuPy: A Python framework for modeling and simulating dynamical systems. *Journal of Open Source Software*, *2*(17), 396. https://doi.org/10.21105/joss.00396

Modelica standard library. (1999). Modelica Association; https://github.com/modelica/ModelicaStandardLibrary.

Sandrock, C. (2019). *A Python library for solving textbook control problems*. https://github.com/alchemyst/Dynamics-and-Control.

Simulink, Simulation and Model-Based Design, Version R2025a. (2025). [Computer software]. The MathWorks, Inc.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2