PySwarms: a research toolkit for Particle Swarm Optimization in Python

Lester James V. Miranda[1]

1 Waseda University

DOI: 10.21105/joss.00433

Software

- Review
- Repository
- Archive

Submitted: 07 October 2017
Published: 10 January 2018

Summary

Particle swarm optimization (PSO) is a heuristic search technique that iteratively improves a set of candidate solutions given an objective measure of fitness (Kennedy and Eberhart 1995b). Although vanilla implementations of PSO can be found in some Python evolutionary algorithm toolboxes (Fortin et al. 2012; Biscani, Izzo, and Märtens 2017), a PSO-specific library that focuses on the said technique is still an open challenge.

PySwarms is a research toolkit for Particle Swarm Optimization (PSO) that provides a set of class primitives useful for solving continuous and combinatorial optimization problems. It follows a black-box approach, solving optimization tasks with few lines of code, yet allows a white-box framework with a consistent API for rapid prototyping of non-standard swarm models. In addition, benchmark objective functions and parameter-search tools are included to evaluate and improve swarm performance. It is intended for swarm intelligence researchers, practitioners, and students who would like a high-level declarative interface for implementing PSO in their problems.

The main design principle of the package is to balance (1) ease-of-use by providing a rich set of classes to solve optimization tasks, and (2) ease-of-experimentation by defining a consistent API to accommodate non-standard PSO implementations. In this context, PySwarms follows these core principles in its development:

- **Maintain a specific set of conventions that are manageable to understand.** This enables repeatability in all implementations and creates a single framework where the API will be based. Thus, for a particular swarm $\mathcal{S}$, the particles are defined as an $m \times n$ matrix where m corresponds to the number of particles, and n to the number of dimensions in the search-space. Its fitness is then expressed as an m-dimensional array containing the value for each particle.
- **Define a consistent API for all swarm implementations.** A consistent API accommodates rapid prototyping of non-standard PSO implementations. As long as the user implements according to the API, all PySwarms capabilities are made available. It consists of an `init` method to initialize the swarm, an `update_position` and `update_velocity` rule to define update behaviour, and an `optimize` method that contains the evolutionary loop.
- **Provide a set of primitive classes for off-the-shelf PSO implementations.** To deliver easy-access of PSO implementations for common optimization tasks, wrapper classes for standard global-best and local-best PSO are included. These implementations follow the same PySwarms API, and can even be built upon for more advanced applications.

Various features include:

- **Python implementation of standard PSO algorithms.** Includes the classic global best and local best PSO (Kennedy and Eberhart 1995b, 1995a), and binary

PSO for discrete optimization (Kennedy and Eberhart 1997). These implementations are built natively in `numpy` (Walt, Colbert, and Varoquaux 2011; Jones et al. 2001–2001--).

- **Built-in single objective functions for testing.** Provides an array of single-objective functions to test optimizers. Includes simple variants such as the sphere function, up to complicated ones such as Beale and Rastrigin functions.
- **Plotting environment for cost and swarm animation.** A wrapper built on top of `matplotlib` (Hunter 2007) to conveniently plot costs and animate swarms (both in 2D and 3D) to assess performance and behavior.
- **Hyperparameter search tools.** Implements both random and grid search to find optimal hyperparameters for controlling swarm behavior.
- **Base classes for implementing your own optimizer.** Provides single-objective base classes for researchers to rapidly prototype and implement their own optimizers.

Example use-cases involve: optimization of continuous and discrete functions, neural network training, feature selection, forward kinematics, and the like. Some of these use-cases are explained, with accompanying code, in the Documentation. This package is actively maintained and developed by the author with the help of various contributors.

References

Biscani, Francesco, Dario Izzo, and Marcus Märtens. 2017. "Esa/Pagmo2: Pagmo 2.6." https://doi.org/10.5281/zenodo.1054110.

Fortin, Félix-Antoine, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. "DEAP: Evolutionary Algorithms Made Easy." *Journal of Machine Learning Research* 13 (July):2171–5.

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.

Jones, Eric, Travis Oliphant, Pearu Peterson, and others. 2001–2001--. "SciPy: Open Source Scientific Tools for Python." http://www.scipy.org/.

Kennedy, James, and Russell Eberhart. 1995a. "A New Optimizer for Particle Swarm Theory." In *Proceedings of the Sixth International Symposium on Micromachine and Human Science.*

———. 1995b. "Particle Swarm Optimization." In *Proceedings of the Ieee International Joint Conference on Neural Networks.*

———. 1997. "A Discrete Binary Particle Swarm Optimization Algorithm." In *Proceedings of the Ieee International Conference on Systems, Man, and Cybernetics.*

Walt, Stefan van der, S. Chris Colbert, and Gael Varoquaux. 2011. "The Numpy Array: A Structure for Efficient Numerical Computation." *Computing in Science & Engineering* 13 (2):22–30. https://doi.org/10.1109/MCSE.2011.37.