GLOBe: A Modular Global Optimization Library

Gaëtan Serré[1], Argyris Kalogeratos[1], Nicolas Vayatis[1]

[1] Centre Borelli, École Normale Supérieure Paris-Saclay, France

Continuous global optimization

Definition 1 – *Problem*

Given $f : \Omega \to \mathbb{R}$ continuous on a compact $\Omega \subset \mathbb{R}^d$, find

$$x^\star \in \arg\max_{x\in\Omega} f(x) = \arg\min_{x\in\Omega}(-f(x)).$$

Outside the convex case, this is **not** local optimization: the space must be explored, usually *stochastically*, and often with f available only as a **black box**. Recent particle-based methods keep the field very active.

Limitations of existing libraries

Library	Scope and limitation
NLopt	Classical algorithms only, no structural factorization
Scipy.optimize	Global optimization secondary to local methods
BayesianOpt	Expensive black-box functions only
Optuna, Ray Tune	Hyperparameter tuning, not general optimization

Across the board: **specialization over generality**, **no factorization of shared patterns** (code duplication, costly extension), and **little coverage of recent advances** — precisely the baselines a new method must be compared against.

The GLOBe library

14	19	C++	Python
optimizers	analytical benchmarks	Eigen backend	user-facing API

A **modular, general-purpose** framework for continuous, possibly black-box, global optimization that:

i. consolidates **classical and state-of-the-art** algorithms, so new methods can be benchmarked against up-to-date baselines;
ii. ships an **integrated benchmarking toolkit** for direct, reproducible comparison;
iii. **factors common patterns at the family level**, turning the structural invariants exposed by recent mathematical formalizations of global optimization into reusable software components.

Modular architecture

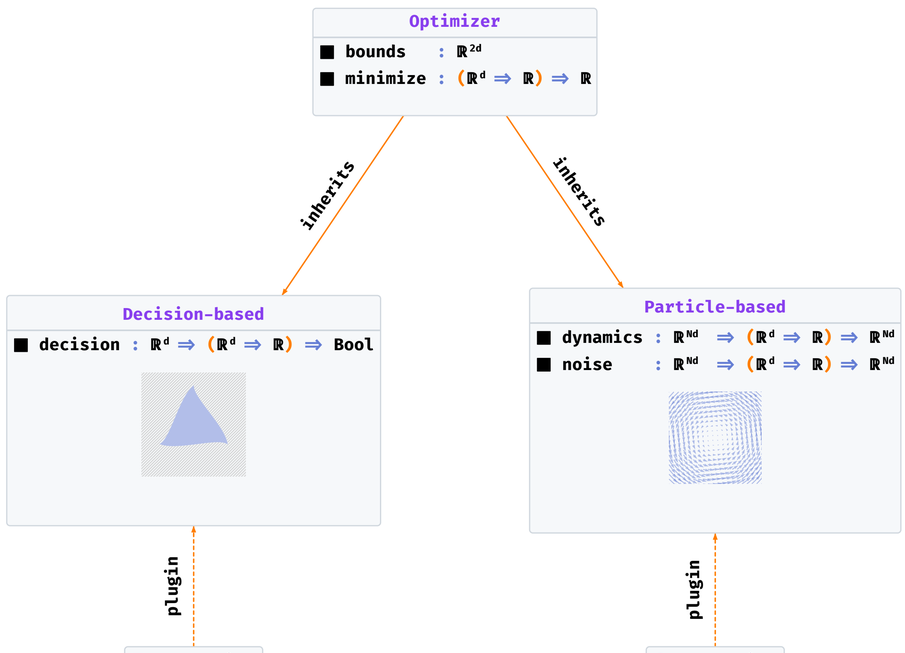

Figure 1. *Class hierarchy of GLOBe. Two family base classes factor the shared iteration logic; orthogonal plugins attach to a family and are inherited by all of its algorithms.*

Every algorithm derives from one abstract class, which already handles bounds, seeded randomness, stopping criteria and the C++/Python interface layer — a new algorithm only implements its own logic.

C++

```cpp
class Optimizer {
public:
  Optimizer(vec_bounds bounds, string name);
  virtual result_eigen minimize(
    function<double(dyn_vector)> f) = 0;
  void set_stop_criterion(double criterion);

  vec_bounds bounds;
  mt19937_64 re;  // Seeded random engine
  double stop_criterion;
};
```

Algorithm families

Decision-based methods decide *where to sample next* from the evaluation history: at step t, turn $\{(x_i, f(x_i))\}_{i<t}$ into a region $R_t \subseteq \Omega$, then draw and evaluate $x_t \sim \mathcal{U}(R_t)$.

Particle-based methods evolve N particles under stochastic dynamics, most of them discretizations of a McKean–Vlasov SDE:

$$\mathrm{d}X_t^{(i)} = b\Big(X_t^{(i)}, \hat{\mu}_t\Big)\,\mathrm{d}t + \sigma\Big(X_t^{(i)}, \hat{\mu}_t\Big)\,\mathrm{d}B_t^{(i)},$$

with drift b, diffusion $\sigma\,\mathrm{d}B_t^{(i)}$ and $\hat{\mu}_t$ the empirical law of the particles. An algorithm supplies only b and σ; the base class runs the Euler–Maruyama loop, boundary handling and scheduling.

Algorithm	Family	Reference
AdaLIPO+	Decision-based	Serré et al., 2024
ECP (Every Call is Precious)	Decision-based	Fourati et al., 2025
AdaRankOpt	Decision-based	Malherbe et al., 2016
PSO (Particle Swarm)	Particle-based	Kennedy et al., 1995
CBO (Consensus-Based Opt.)	Particle-based	Pinnau et al., 2017
Langevin dynamics	Particle-based	Prigogine, 1957
SBS (Stein Boltzmann Sampling)	Particle-based	Serré et al., 2025
MSGD (Multi-Start GD)	Particle-based	–
DIRECT	Misc.	Jones, 2001
CRS (Controlled Random Search)	Misc.	Price, 1983
CMA-ES	Misc.	Hansen et al., 1996
MLSL (Multi-Level Single-Linkage)	Misc.	Rinnooy Kan et al., 1985
Gradient Descent, Pure Random Search	Misc.	–

Table 1. *The 14 optimizers shipped with GLOBe.*

Family-level plugins

Written once against a family base class, **valid for every algorithm of that family**.

- **Trust region** (decision-based) — on a promising region, switch temporarily to a local optimizer (BOBYQA), then resume global exploration.
- **Common noise** (particle-based) — independent noise gives too little trajectory diversity; noise *shared* across particles restores exploration. *SMD* perturbs the aggregate statistics, *GCN* builds a canonical Gaussian common noise from the process geometry.
- **Particle filtering** (particle-based) — quantile filtering drops particles with small displacement and poor values; new strategies (e.g. SIR) plug in the same way.

Benchmarking toolkit

19 analytical functions, in *any* dimension, most with known global minima:

- *Unimodal* (9): Sphere, Ellipsoid, Bentcigar, Hyperellipsoid, Zakharov, Trid, Sumpow, Rosenbrock, Dixonprice.
- *Multimodal* (10): Rastrigin, Ackley, Schwefel, Levy, Michalewicz, Langermann, Deb, Griewank, Styblinski-Tang.

PyGKLS, wrapping the GKLS *random* function generator: known global and local minima, controllable multi-modality and smoothness (C^2 to C^0), seeded — statistics over random instances, not a handful of fixed functions.

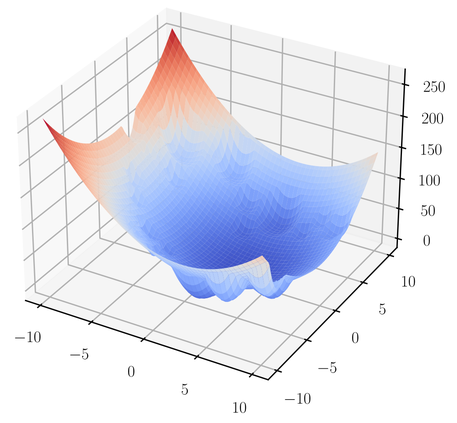

Figure 2. *A random GKLS function.*

The **GLOBe** class runs every optimizer on every benchmark n times, then aggregates the requested metrics into console or LaTeX tables.

Python

```python
from globe import GLOBe
from globe.benchmarks import PyGKLS, create_bounds

tr = {"radius": 0.1, "bobyqa_eval": 20}  # trust region
globe = GLOBe(
  optimizers=[
    ("CBO", {"n_particles": 50, "filter_type": "quantile"}),
    ("SMD-CBO", {"n_particles": 50}),   # + common noise
    ("AdaLIPO", {"trust_region_dict": tr}),
  ],
  benchmarks=[PyGKLS(dim, 15, [-100, 100], -100)],
  metrics=["Proportion"],
  bounds=[create_bounds(dim, -99, 99)],
)
globe.run(n_runs=50, latex_table=True)
```

Extensible metrics. Each metric is a callable over the solutions of all runs. Built in: mean and standard deviation of the values and the **success rate**, the fraction of runs below f_{target}, where $f_{\text{target}} = f_{\min} + (f_{\text{mean}} - f_{\min})(1-p)$, with f_{mean} the mean of f over a large random sample of Ω. Custom metrics subclass **Metric**.

14 algorithms, 19+ benchmarks, plugins written once that work family-wide.