# Capabilities

## Contents



## Overview

- Support for a wide array of types of optimisation problems (continuous, integer, single and multi-objective, constrained and unconstrained, with or without derivatives, stochastic, etc.).
- A comprehensive library of algorithms, including global and local solvers, meta-heuristics, single and multi-objective algorithms, wrappers for third-party solvers (e.g., NLopt, Ipopt, etc.).
- Comprehensive support for coarse-grained parallelisation via the generalised island model. In the island model, multiple optimisation instances run in parallel (possibly on different machines) and exchange information as the optimisation proceeds, improving the overall time-to-solution and allowing to harness the computational power of modern computer architectures (including massively-parallel high-performance clusters).
- Support for fine-grained parallelisation (i.e., at the level of single objective function evaluations) in selected algorithms via the batch fitness evaluation framework. This allows to speed-up single optimisations via parallel processing (e.g., multithreading, high-performance clusters, GPUs, SIMD vectorization, etc.).
- A library of ready-to-use optimisation problems for algorithmic testing and performance evaluation (Rosenbrock, Rastrigin, Lennard-Jones, etc.).
- A library of optimisation-oriented utilities (e.g., hypervolume computation, non-

Skip to main content

# List of algorithms

This is the list of user defined algorithms (UDAs) currently provided with pagmo. These are classes that can be used to construct a `pagmo::algorithm`, which will then provide a unified interface to access the algorithm's functionalities.

Generally speaking, algorithms can solve only specific problem classes. In the tables below, we use the following flags to signal which problem types an algorithm can solve:

- S = Single-objective
- M = Multi-objective
- C = Constrained
- U = Unconstrained
- I = Integer programming
- sto = Stochastic

Note that algorithms that do not directly support integer programming will still work on integer problems (i.e., they will optimise the relaxed problem). Note also that it is possible to use meta-problems to turn constrained problems into unconstrained ones, and multi-objective problems into single-objective ones.

Skip to main content

# Heuristic Global Optimization

| Common Name | Docs of the C++ class | Capabilities |
|---|---|---|
| Extended Ant Colony Optimization (GACO) | `pagmo::gaco` | S-CU-I |
| Differential Evolution (DE) | `pagmo::de` | S-U |
| Self-adaptive DE (jDE and iDE) | `pagmo::sade` | S-U |
| Self-adaptive DE (de_1220 aka pDE) | `pagmo::de1220` | S-U |
| Grey wolf optimizer (GWO) | `pagmo::gwo` | S-U |
| Improved Harmony Search | `pagmo::ihs` | SM-CU-I |
| Particle Swarm Optimization (PSO) | `pagmo::pso` | S-U |
| Particle Swarm Optimization Generational (GPSO) | `pagmo::pso_gen` | S-U-sto |
| (N+1)-ES Simple Evolutionary Algorithm | `pagmo::sea` | S-U-sto |
| Simple Genetic Algorithm | `pagmo::sga` | S-U-I-sto |
| Corana's Simulated Annealing (SA) | `pagmo::simulated_annealing` | S-U |
| Artificial Bee Colony (ABC) | `pagmo::bee_colony` | S-U |
| Covariance Matrix Adaptation Evo. Strategy (CMA-ES) | `pagmo::cmaes` | S-U-sto |
| Exponential Evolution Strategies (xNES) | `pagmo::xnes` | S-U-sto |
| Non-dominated Sorting GA (NSGA2) | `pagmo::nsga2` | M-U-I |
| Multi-objective EA with Decomposition (MOEA/D) | `pagmo::moead` | M-U |
| Multi-objective EA with Decomposition Generational (GMOEA/D) | `pagmo::moead_gen` | M-U |
| Multi-objective Hypervolume-based | `pagmo::maco` | M-U-I |

Skip to main content

| Common Name | Docs of the C++ class | Capabilities |
| --- | --- | --- |
| Non-dominated Sorting PSO (NSPSO) | `pagmo::nspso` | M-U |

Skip to main content

# Local optimization

| Common Name | Docs of the C++ class | Capabilities |
|---|---|---|
| Compass Search (CS) | `pagmo::compass_search` | S-CU |
| COBYLA (from NLopt) | `pagmo::nlopt` | S-CU |
| BOBYQA (from NLopt) | `pagmo::nlopt` | S-U |
| NEWUOA + bound constraints (from NLopt) | `pagmo::nlopt` | S-U |
| PRAXIS (from NLopt) | `pagmo::nlopt` | S-U |
| Nelder-Mead simplex (from NLopt) | `pagmo::nlopt` | S-U |
| Subplex (from NLopt) | `pagmo::nlopt` | S-U |
| MMA (Method of Moving Asymptotes) (from NLopt) | `pagmo::nlopt` | S-CU |
| CCSA (from NLopt) | `pagmo::nlopt` | S-CU |
| SLSQP (from NLopt) | `pagmo::nlopt` | S-CU |
| Low-storage BFGS (from NLopt) | `pagmo::nlopt` | S-U |
| Preconditioned truncated Newton (from NLopt) | `pagmo::nlopt` | S-U |
| Shifted limited-memory variable-metric (from NLopt) | `pagmo::nlopt` | S-U |
| Ipopt | `pagmo::ipopt` | S-CU |
| SNOPT (in pagmo_plugins_non_free affiliated package) | pagmo::snopt7 | S-CU |
| WORHP (in | pagmo::worhp | S-CU |

Skip to main content

| Common Name | Docs of the C++ class | Capabilities |
|---|---|---|
| affiliated package) | | |

## Meta-algorithms

| Common Name | Docs of the C++ class | Capabilities [1] |
|---|---|---|
| Monotonic Basin Hopping (MBH) | `pagmo::mbh` | S-CU |
| Cstrs Self-Adaptive | `pagmo::cstrs_self_adaptive` | S-C |
| Augmented Lagrangian algorithm (from NLopt) [2] | `pagmo::nlopt` | S-CU |

**Footnotes**

[1] The capabilities of the meta-algorithms depend also on the capabilities of the algorithms they wrap. If, for instance, a meta-algorithm supporting constrained problems is constructed from an algorithm which does *not* support constrained problems, the resulting meta-algorithms will *not* be able to solve constrained problems.

[2] The Augmented Lagrangian algorithm can be used only in conjunction with other NLopt algorithms.

# List of problems

This is the list of user defined problems (UDPs) currently provided with pagmo. These are classes that can be used to construct a `pagmo::problem`, which will then provide a unified interface to access the problem's functionalities.

In the tables below, we classify optimisation problems according to the following flags:

- S = Single-objective
- M = Multi-objective
- C = Constrained
- U = Unconstrained
- I = Integer programming

Skip to main content

- sto = Stochastic

## Scalable problems

| Common Name | Docs of the C++ class | Type |
|---|---|---|
| Ackley | `pagmo::ackley` | S-U |
| Golomb Ruler | `pagmo::golomb_ruler` | S-C-I |
| Griewank | `pagmo::griewank` | S-U |
| Hock Schittkowski 71 | `pagmo::hock_schittkowski_71` | S-C |
| Inventory | `pagmo::inventory` | S-U-sto |
| Lennard Jones | `pagmo::lennard_jones` | S-U |
| Luksan Vlcek 1 | `pagmo::luksan_vlcek1` | S-C |
| Rastrigin | `pagmo::rastrigin` | S-U |
| MINLP Rastrigin | `pagmo::minlp_rastrigin` | S-U-I |
| Rosenbrock | `pagmo::rosenbrock` | S-U |
| Schwefel | `pagmo::schwefel` | S-U |

Skip to main content

## Problem suites

| Common Name | Docs of the C++ class | Type |
|---|---|---|
| CEC2006 | `pagmo::cec2006` | S-C |
| CEC2009 | `pagmo::cec2009` | S-C |
| CEC2013 | `pagmo::cec2013` | S-U |
| CEC2014 | `pagmo::cec2014` | S-U |
| ZDT | `pagmo::zdt` | M-U |
| DTLZ | `pagmo::dtlz` | M-U |
| WFG | `pagmo::wfg` | M-U |

## Meta-problems

Meta-problems are UDPs that take another UDP as input, yielding a new UDP which modifies the behaviour and/or the properties of the original problem in a variety of ways.

| Common Name | Docs of the C++ class |
|---|---|
| Decompose | `pagmo::decompose` |
| Translate | `pagmo::translate` |
| Unconstrain | `pagmo::unconstrain` |

# List of islands

This is the list of user defined islands (UDIs) currently provided with pagmo. These are classes that can be used to construct a `pagmo::island`, which will then provide a unified interface to access the island's functionalities.

In the pagmo jargon, an island is an entity tasked with managing the asynchronous evolution of a population via an algorithm in the generalised island model. Different UDIs enable different parallelisation strategies (e.g., multithreading, multiprocessing, cluster

Skip to main content

| Common Name | Docs of the C++ class |
| --- | --- |
| Thread island | `pagmo::thread_island` |
| Fork island | `pagmo::fork_island` |

# List of batch fitness evaluators

This is the list of user defined batch fitness evaluators (UDBFEs) currently provided with pagmo. These are classes that can be used to construct a `pagmo::bfe`, which will then provide a unified interface to access the evaluator's functionalities.

In the pagmo jargon, a batch fitness evaluator implements the capability of evaluating a group of decision vectors in a parallel and/or vectorised fashion. Batch fitness evaluators are used to implement fine-grained parallelisation in pagmo (e.g., parallel initialisation of populations, or parallel fitness evaluations within the inner loop of an algorithm).

| Common Name | Docs of the C++ class |
| --- | --- |
| Default BFE | `pagmo::default_bfe` |
| Thread BFE | `pagmo::thread_bfe` |