

Package: optimix (via r-universe)

October 10, 2026

Title A Unified Contract and Auto-Selector for Efficient Optimisers

Version 0.2.0

Description Provides a single typed problem contract over a curated portfolio of efficient optimisers, with an automatic per-instance selector that picks the best optimiser for a given problem and returns it in an 'stats::optim()'-compatible result object. The base solver runs with no suggested packages installed; each additional engine, and an optional mixture-valued mapper from 'proxymix', is loaded lazily only when used.

License MIT + file LICENSE

Encoding UTF-8

Language en-AU

Depends R (>= 4.5)

Imports graphics, S7, stats, utils

Suggests cmaes, DEoptim, DEoptimR, dfoptim, DiceKriging, GenSA, ggplot2, knitr, nloptr, proxymix, rmarkdown, smoof, testthat (>= 3.0.0)

VignetteBuilder knitr

URL <https://github.com/max578/optimix>

BugReports <https://github.com/max578/optimix/issues>

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Repository <https://max578.r-universe.dev>

Date/Publication 2026-10-09 22:29:28 UTC

RemoteUrl <https://github.com/max578/optimix>

RemoteRef main

RemoteSha dcad718cafc9890caaaa5ed614c4ba213488578b

Contents

as_optim	2
as_orchestra_manifest	3
list_optimisers	4
objective_deterministic	4
objective_expensive	5
objective_noisy	5
objective_spec	6
opt_space	7
optim_engine	7
optim_problem	8
optimix	9
optimix_map	11
orchestra_manifest	12
register_optimiser	13
space_box	14
space_permutation	14
verify_manifest	15
Index	16

as_optim	<i>Coerce an optimisation result to a base optim() list</i>
----------	---

Description

Returns the bare stats::optim()-shaped components of a result, for code paths that expect exactly that list and nothing more.

Usage

```
as_optim(x, ...)
```

Arguments

- x An object to coerce.
- ... Unused; for method consistency.

Value

A list with par, value, counts, convergence, and message.

See Also

Other results: [optimix\(\)](#), [optimix_map\(\)](#)

Examples

```
res <- optimix(function(x) sum(x^2), c(-5, -5), c(5, 5),
               method = "base_optim")
as_optim(res)
```

as_orchestra_manifest *Emit an optimix result as an orchestra manifest*

Description

The federation's emit / adapt generic. The `optimix_result` method turns an optimisation result into the shared `orchestra_manifest` contract: `inferential_target = "parameters"`, the optimum carried as a one-row `params` table (the best parameter vector and its objective value), and the engine provenance – which optimiser ran, why it was chosen, convergence, the true evaluation count, and whether an uncertainty quantification is available – in the typed summary and metadata. When the proxymix mixture engine ran, `metadata$solution_map` carries the queryable mixture over all optima, so a downstream consumer that needs a posterior over the optimum (not just a point) can read it; a point engine sets `metadata$uncertainty_available = FALSE`. The payload is integrity-hashed so a consumer can verify it with [verify_manifest\(\)](#).

Usage

```
as_orchestra_manifest(x, ...)

## S3 method for class 'optimix_result'
as_orchestra_manifest(x, ..., run_id = NULL, seed = NA_integer_)
```

Arguments

<code>x</code>	An <code>optimix_result</code> object.
<code>...</code>	Further arguments passed to methods.
<code>run_id</code>	Optional character run identifier; a content hash is derived when omitted, so two identical optima get the same identifier.
<code>seed</code>	Optional integer recorded for reproducibility.

Value

An `orchestra_manifest` S7 object.

See Also

[verify_manifest\(\)](#), [optimix\(\)](#)

Other orchestra manifest: [orchestra_manifest\(\)](#), [verify_manifest\(\)](#)

Examples

```
res <- optimix(function(x) sum((x - 0.5)^2), c(-2, -2), c(2, 2),
              method = "base_optim")
as_orchestra_manifest(res)
```

list_optimisers	<i>List the registered optimisers</i>
-----------------	---------------------------------------

Description

Reports every registered engine, whether its backend package is installed, and the metadata method = "auto" uses to choose between them.

Usage

```
list_optimisers()
```

Value

A data frame, one row per engine, with columns name, package, installed, global, noise_tolerant, and emits_map.

See Also

Other engines and registry: [optim_engine\(\)](#), [register_optimiser\(\)](#)

Examples

```
list_optimisers()
```

objective_deterministic	<i>Declare a deterministic objective</i>
-------------------------	--

Description

Marks the objective as deterministic, so repeated evaluations at the same point return the same value. This is the default for [optim_problem\(\)](#).

Usage

```
objective_deterministic()
```

Value

An [objective_spec](#) with kind = "deterministic".

See Also

Other objective declarations: [objective_expensive\(\)](#), [objective_noisy\(\)](#), [objective_spec\(\)](#)

Examples

```
objective_deterministic()
```

objective_expensive	<i>Declare an expensive objective</i>
---------------------	---------------------------------------

Description

Marks the objective as expensive to evaluate, so the selector prefers a surrogate-model (Bayesian-optimisation) engine that spends few evaluations.

Usage

```
objective_expensive()
```

Value

An [objective_spec](#) with kind = "expensive".

See Also

Other objective declarations: [objective_deterministic\(\)](#), [objective_noisy\(\)](#), [objective_spec\(\)](#)

Examples

```
objective_expensive()
```

objective_noisy	<i>Declare a noisy objective</i>
-----------------	----------------------------------

Description

Marks the objective as stochastic, so the selector prefers a noise-tolerant optimiser.

Usage

```
objective_noisy(sd = NA_real_)
```

Arguments

sd	A single non-negative number, an optional estimate of the observation-noise standard deviation. Use NA when it is unknown.
----	--

Value

An `objective_spec` with `kind = "noisy"`.

See Also

Other objective declarations: `objective_deterministic()`, `objective_expensive()`, `objective_spec()`

Examples

```
objective_noisy(sd = 0.1)
```

<code>objective_spec</code>	<i>Objective specification</i>
-----------------------------	--------------------------------

Description

Describes the nature of the objective so the selector and the engines can treat it appropriately. Construct one with `objective_deterministic()` or `objective_noisy()` rather than calling the class directly.

Usage

```
objective_spec(kind = "deterministic", noise_sd = NA_real_)
```

Arguments

<code>kind</code>	A single string: "deterministic", "noisy", or "expensive".
<code>noise_sd</code>	A single number, an optional estimate of the observation noise standard deviation, or NA when unknown.

Value

An S7 object of class `objective_spec`.

See Also

Other objective declarations: `objective_deterministic()`, `objective_expensive()`, `objective_noisy()`

Examples

```
objective_spec(kind = "noisy", noise_sd = 0.1)
```

opt_space	<i>Design-space base class</i>
-----------	--------------------------------

Description

Abstract parent for the design spaces an [optim_problem\(\)](#) can search. Each concrete space declares the geometry an optimiser must respect. The concrete spaces in this release are [space_box\(\)](#) and [space_permutation\(\)](#).

Usage

```
opt_space()
```

Value

Not constructed directly; see [space_box\(\)](#).

See Also

Other design spaces: [optim_problem\(\)](#), [space_box\(\)](#), [space_permutation\(\)](#)

optim_engine	<i>An optimiser engine specification</i>
--------------	--

Description

Bundles the metadata that drives selection with the adapter that maps a [optim_problem](#) to a backend optimiser and returns an [optimix_result](#). The `accepts`, `global`, `noise_tolerant`, `emits_map`, and `dimension` fields are exactly what `method = "auto"` reads when it chooses an engine, so registering an engine and teaching the selector about it are one and the same act.

Usage

```
optim_engine(
  name = character(0),
  pkg = "base",
  accepts = "space_box",
  global = FALSE,
  uses_gradient = FALSE,
  noise_tolerant = FALSE,
  stochastic = FALSE,
  emits_map = FALSE,
  dim_min = 1,
  dim_max = Inf,
  available = function() NULL,
  run = function() NULL
)
```

Arguments

name	A single string naming the engine; this is the value passed to method <code>=</code> .
pkg	A single string naming the package the engine needs, or "base" when it needs nothing beyond base R.
accepts	A character vector of the design-space class names the engine can search, such as "space_box".
global	A single logical, TRUE if the engine performs global search.
uses_gradient	A single logical, TRUE if the engine uses gradients.
noise_tolerant	A single logical, TRUE if the engine handles noisy objectives well.
stochastic	A single logical, TRUE if the engine is randomised and therefore honours a seed.
emits_map	A single logical, TRUE if the engine returns a mixture-valued solution map.
dim_min, dim_max	Single numbers bounding the problem dimensions the engine supports.
available	A function of no arguments returning a single logical: is the engine usable in the current session?
run	A function taking an optim_problem and returning an <code>optimix_result</code> .

Value

An S7 object of class `optim_engine`.

See Also

Other engines and registry: [list_optimisers\(\)](#), [register_optimiser\(\)](#)

Examples

```
optim_engine(
  name = "my_solver",
  pkg = "base",
  available = function() TRUE,
  run = function(problem) NULL
)
```

optim_problem

A unified optimisation problem

Description

The typed contract that every `optimix` engine consumes. The easy facade [optimix\(\)](#) builds one of these from `fn`, `lower`, and `upper`; construct it directly to set a noisy objective, a warm start, an evaluation budget, or a seed.

Arguments

fn	The objective function. It takes a single numeric vector – a point in the design space – and returns a single finite number to be minimised (or maximised when <code>maximise = TRUE</code>).
space	The design space to search, such as a space_box() .
objective	An objective_spec describing the objective; defaults to objective_deterministic() .
maximise	A single logical. When TRUE the objective is maximised rather than minimised. Defaults to FALSE.
max_evals	A single number, the soft budget of objective evaluations, or NA to let each engine use its own default.
warm_start	A numeric vector to start the search from, or a zero-length vector for none. When supplied it must match the space dimension.
seed	A single integer seed for reproducible stochastic search, or NA for none.
delta_fn	An optional incremental-scoring function with signature <code>function(perm, i, j)</code> returning the change in the objective when the permutation entries at positions <code>i</code> and <code>j</code> are swapped. When supplied, a combinatorial engine uses it for fast incremental scoring instead of re-evaluating <code>fn</code> . NULL (the default) means none.

Value

An S7 object of class `optim_problem`.

See Also

Other design spaces: [opt_space\(\)](#), [space_box\(\)](#), [space_permutation\(\)](#)

Examples

```
optim_problem(
  fn = function(x) sum(x^2),
  space = space_box(lower = c(-5, -5), upper = c(5, 5))
)
```

 optimix

Optimise a function

Description

The single entry point. Pass a function with lower and upper bounds for the easy path, or a prepared [optim_problem\(\)](#) for full control. With `method = "auto"` (the default) the best available optimiser is chosen for the problem and recorded in the result's provenance.

Usage

```

optimix(
  fn,
  lower = NULL,
  upper = NULL,
  ...,
  method = "auto",
  maximise = FALSE,
  max_evals = NULL,
  goal = c("optimum", "map")
)

minimise(fn, lower, upper, ...)

maximise(fn, lower, upper, ...)

```

Arguments

<code>fn</code>	The objective function, taking a numeric vector and returning a single finite number; or a prepared optim_problem() , in which case <code>lower</code> , <code>upper</code> , <code>maximise</code> , and <code>max_evals</code> are ignored.
<code>lower</code>	A numeric vector of lower bounds (easy path).
<code>upper</code>	A numeric vector of upper bounds (easy path).
<code>...</code>	Further arguments passed on to <code>fn</code> at each evaluation.
<code>method</code>	The optimiser to use: "auto" (the default) to select one per instance, "race" to race the installed global engines and commit to the leader, or the name of a registered engine (see list_optimisers()).
<code>maximise</code>	A single logical; maximise rather than minimise. Defaults to FALSE.
<code>max_evals</code>	A soft budget of objective evaluations, or NULL to let the engine use its own default.
<code>goal</code>	What to return: "optimum" (the default) for the single best point, or "map" for the full set of optima with a posterior over the optimum-set. <code>goal = "map"</code> steers <code>method = "auto"</code> to a map-emitting engine (proxymix's <code>from_objective_mixture</code>); it is ignored when a specific method is named. For the richer queryable map object see optimix_map() .

Value

An `optimix_result`: a list shaped like a [stats::optim\(\)](#) result (`par`, `value`, `counts`, `convergence`, `message`) with optimix extras (`provenance`, `map`, `diagnostics`, `problem`). It has `print()`, `summary()`, `plot()`, `coef()`, `as.data.frame()`, and `as_optim()` methods. Results embed the problem, including the objective closure and its environment, and are therefore session objects; `as_optim()` gives the minimal durable form for storage.

See Also

Other results: [as_optim\(\)](#), [optimix_map\(\)](#)

Examples

```
# Easy path: minimise a quadratic on a box.
optimix(function(x) sum(x^2), lower = c(-5, -5), upper = c(5, 5),
        method = "base_optim")

# Power path: a prepared problem.
prob <- optim_problem(
  fn = function(x) sum((x - 1)^2),
  space = space_box(lower = c(-5, -5), upper = c(5, 5))
)
optimix(prob, method = "base_optim")
```

optimix_map

*Map the optima of a function***Description**

Returns the distinct optima of `fn` over a box, rather than a single `argmin`. Natively it runs a local search from many starts and clusters the results; when a new-enough `proxymix` is installed it returns a calibrated Gaussian-mixture map over the optima instead.

Usage

```
optimix_map(
  fn,
  lower,
  upper,
  ...,
  n_starts = NULL,
  tol = 0.05,
  max_evals = NULL
)
```

Arguments

<code>fn</code>	The objective function: takes a numeric vector, returns a single finite number to be minimised.
<code>lower, upper</code>	Numeric vectors of box bounds, of equal length.
<code>...</code>	Further arguments passed on to <code>fn</code> .
<code>n_starts</code>	The number of local-search starts (native path); defaults to <code>max(20, 10 * dimension)</code> .
<code>tol</code>	The merge distance for clustering optima, as a fraction of the box width. Defaults to <code>0.05</code> .
<code>max_evals</code>	The per-start local-search budget, or <code>NULL</code> for the default.

Value

An `optimix_map`: a list with modes (a matrix, one optimum per row, best first), values, n, source ("native" or "proxymix"), and, on the proxymix path, map (the queryable mixture).

See Also

Other results: [as_optim\(\)](#), [optimix\(\)](#)

Examples

```
# A double well with two optima at (+/-1, 0).
optimix_map(
  function(x) (x[1]^2 - 1)^2 + x[2]^2,
  lower = c(-2, -2), upper = c(2, 2)
)
```

orchestra_manifest	<i>The orchestra ensemble-manifest contract (optimix-side implementation)</i>
--------------------	---

Description

A versioned, hashed, provenance-complete S7 result object, property-compatible with the federation's reference `orchestra_manifest`. An `optimix_result` is adapted into one through [as_orchestra_manifest\(\)](#): the optimum rides in params (`inferential_target = "parameters"`), the engine, convergence and uncertainty-availability ride in metadata and the typed summary, and the payload is integrity-hashed so a tampered optimum is detected downstream.

Value

An S7 object of class `orchestra_manifest`.

See Also

[as_orchestra_manifest\(\)](#), [verify_manifest\(\)](#)

Other orchestra manifest: [as_orchestra_manifest\(\)](#), [verify_manifest\(\)](#)

Examples

```
# A minimal hand-built manifest; as_orchestra_manifest() fills these
# slots (and a real payload hash) from an optimix result.
m <- orchestra_manifest(
  emitter_package = "optimix",
  emitter_version = "0.0.0.9000",
  inferential_target = "parameters",
  run_id = "example-0001",
  method = "optimix:base_optim",
  params = data.frame(par1 = 0.5, value = 0.25),
```

```
    timestamp = Sys.time(),  
    data_hash = "sha256:unhashed-example"  
  )  
  m@run_id
```

register_optimiser	<i>Register an optimiser engine</i>
--------------------	-------------------------------------

Description

Adds an [optim_engine](#) to the registry so it can be selected by name or by method = "auto". Registering an engine with the same name as an existing one replaces it; replacing one of the built-in engines additionally emits a warning, since it changes what method = "auto" can dispatch to.

Usage

```
register_optimiser(engine)
```

Arguments

engine	An optim_engine to register.
--------	--

Value

The engine name, invisibly.

See Also

Other engines and registry: [list_optimisers\(\)](#), [optim_engine\(\)](#)

Examples

```
engine <- optim_engine(  
  name = "my_solver",  
  pkg = "base",  
  available = function() TRUE,  
  run = function(problem) NULL  
)  
register_optimiser(engine)  
"my_solver" %in% list_optimisers()$name
```

space_box	<i>Continuous box-constrained design space</i>
-----------	--

Description

A continuous, real-valued design space bounded below and above by vectors of equal length. This is the design space the easy facade `optimix()` constructs from its lower and upper arguments.

Usage

```
space_box(lower = integer(0), upper = integer(0))
```

Arguments

lower	A numeric vector of lower bounds, one per dimension. Every element must be finite and not greater than its upper counterpart.
upper	A numeric vector of upper bounds, the same length as lower.

Value

An S7 object of class `space_box`.

See Also

Other design spaces: `opt_space()`, `optim_problem()`, `space_permutation()`

Examples

```
space_box(lower = c(-5, -5), upper = c(5, 5))
```

space_permutation	<i>Permutation design space</i>
-------------------	---------------------------------

Description

A combinatorial design space whose points are permutations of `seq_len(n)`. Use it for ordering and assignment problems via the powerful path: `optim_problem(fn, space = space_permutation(n))`, where `fn` takes a permutation (an integer vector).

Usage

```
space_permutation(n = integer(0))
```

Arguments

n	A single integer of at least 2: the number of items to order.
---	---

Value

An S7 object of class space_permutation.

See Also

Other design spaces: [opt_space\(\)](#), [optim_problem\(\)](#), [space_box\(\)](#)

Examples

```
space_permutation(5)
```

verify_manifest	<i>Verify an orchestra manifest's payload integrity</i>
-----------------	---

Description

Recomputes the payload hash from the object's own load-bearing slots and compares it to the stored data_hash. A mismatch means the payload was modified after emission. Matches the reference contract's verify_manifest().

Usage

```
verify_manifest(m)
```

Arguments

m An orchestra_manifest object.

Value

A list with logical ok and a human-readable message.

See Also

[as_orchestra_manifest\(\)](#)

Other orchestra manifest: [as_orchestra_manifest\(\)](#), [orchestra_manifest\(\)](#)

Examples

```
res <- optimix(function(x) sum(x^2), c(-5, -5), c(5, 5),
               method = "base_optim")
m <- as_orchestra_manifest(res)
verify_manifest(m)
```

Index

- * **design spaces**
 - opt_space, 7
 - optim_problem, 8
 - space_box, 14
 - space_permutation, 14
- * **engines and registry**
 - list_optimisers, 4
 - optim_engine, 7
 - register_optimiser, 13
- * **objective declarations**
 - objective_deterministic, 4
 - objective_expensive, 5
 - objective_noisy, 5
 - objective_spec, 6
- * **orchestra manifest**
 - as_orchestra_manifest, 3
 - orchestra_manifest, 12
 - verify_manifest, 15
- * **results**
 - as_optim, 2
 - optimix, 9
 - optimix_map, 11

as_optim, 2

as_optim(), 10, 12

as_orchestra_manifest, 3

as_orchestra_manifest(), 12, 15

list_optimisers, 4

list_optimisers(), 8, 10, 13

maximise(optimix), 9

minimise(optimix), 9

objective_deterministic, 4

objective_deterministic(), 5, 6, 9

objective_expensive, 5

objective_expensive(), 5, 6

objective_noisy, 5

objective_noisy(), 5, 6

objective_spec, 4, 5, 6, 6, 9

objective_spec(), 5, 6

opt_space, 7

opt_space(), 9, 14, 15

optim_engine, 7, 13

optim_engine(), 4, 13

optim_problem, 7, 8, 8

optim_problem(), 4, 7, 9, 10, 14, 15

optimix, 9

optimix(), 2, 3, 8, 12, 14

optimix_map, 11

optimix_map(), 2, 10

optimix_result, 7

orchestra_manifest, 12

orchestra_manifest(), 3, 15

register_optimiser, 13

register_optimiser(), 4, 8

space_box, 14

space_box(), 7, 9, 15

space_permutation, 14

space_permutation(), 7, 9, 14

stats::optim(), 10

verify_manifest, 15

verify_manifest(), 3, 12