

Package: decideR (via r-universe)

October 15, 2026

Title Bayesian Decision-Theoretic Actions under Loss and Hard Constraints

Version 0.3.1.9000

Description Maps a posterior distribution or an 'orchestra_manifest' inference result to an optimal action under an explicit loss function and a hard feasibility constraint, returning the action together with a grounding label and an abstention when evidence is insufficient. The grounding label is combined worst-case across inputs, so a decision resting on an unverified external fact is downgraded to abstention rather than asserted with false confidence. A manifest-native tail closes an analytics pipeline by deciding directly from an upstream member's inference result, and a grade-band loss prices quality-threshold ('protein-band') payoffs as a first-class step value schedule. Reference decisions cover bounded input-rate selection and constraint-capped position sizing.

License MIT + file LICENSE

URL <https://github.com/max578/decideR>,
<https://max578.github.io/decideR>

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

Encoding UTF-8

Language en-AU

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Imports S7, stats

Suggests ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-10-15 23:05:20 UTC
RemoteUrl https://github.com/max578/decideR
RemoteRef main
RemoteSha bf967c2e5af957cbdad188309fc1d1cedba84cef

Contents

as.data.frame.decider_bcr	2
bcr_crrdc	3
combine_grounding	5
decide	6
decide_bcr	8
decide_evpi	10
decide_evsi	12
decide_from_manifest	14
decide_input_rate	16
decide_position_size	17
decide_rate_from_manifest	19
decision	21
grade_band_utility	22
grade_band_value	23
grounding_tokens	24
is_grounded	25
print,decideR::decision-method	26
print,decideR::valuation-method	26
print.decider_bcr	27
print.grade_band_value	27
valuation	28
Index	30

as.data.frame.decider_bcr	<i>Coerce a CRRDC benefit-cost ratio to a data.frame</i>
---------------------------	--

Description

Coerce a CRRDC benefit-cost ratio to a data.frame

Usage

```
## S3 method for class 'decider_bcr'  
as.data.frame(x, ...)
```

Arguments

`x` A `bcr_crrdc()` result.

`...` Ignored, for compatibility with `as.data.frame()`.

Value

The rates data.frame: one row per discount rate, columns `rate_label`, `rate`, `pv_benefits`, `pv_costs`, `npv`, `bcr`.

Examples

```
bens <- data.frame(year = 2020:2021, value = c(0, 210))
csts <- data.frame(year = 2020:2021, value = c(100, 0))
up   <- data.frame(year = 2020:2021, uptake = c(1, 1))
as.data.frame(bcr_crrdc(bens, csts, counterfactual = "already_net",
                        counterfactual_label = "no intervention",
                        attribution = 1, adoption = up,
                        grounding = grounding_grounded()))
```

bcr_crrdc

*CRRDC-guideline benefit-cost ratio***Description**

Discounts a benefit stream and a cost stream to a benefit-cost ratio in the shape a Council of Rural Research and Development Corporations (CRRDC) impact assessment reports, with the four inputs the guidelines make mandatory: an explicit counterfactual, an explicit attribution share, an adoption profile with lags, and a discount-rate sensitivity band around the central rate. Where `decide_bcr()` leaves those assumptions to the caller, `bcr_crrdc()` refuses to construct a ratio without all four – a benefit-cost ratio that implicitly assumes no counterfactual, full attribution and instant 100% uptake is exactly what the guidelines forbid.

Usage

```
bcr_crrdc(
  benefits,
  costs,
  counterfactual,
  attribution,
  adoption,
  counterfactual_label = NULL,
  discount_central = 0.05,
  discount_sensitivity = c(0.03, 0.07),
  base_year = NULL,
  grounding = NULL
)
```

Arguments

benefits	A <code>data.frame(year, value)</code> : the (gross, pre-counterfactual) annual real benefit stream.
costs	A <code>data.frame(year, value)</code> : the annual real cost stream. Not adjusted by counterfactual, attribution or adoption.
counterfactual	Required, no default. Either a <code>data.frame(year, value)</code> of benefits under the counterfactual, spanning every year benefits covers, subtracted from benefits; or the string "already_net" declaring benefits is already incremental, in which case <code>counterfactual_label</code> is also required.
attribution	Required, no default. A single number in $(0, 1]$: the share of the incremental benefit attributable to this investment.
adoption	Required, no default. A <code>data.frame(year, uptake)</code> with uptake in $[0, 1]$, spanning every year benefits covers, applied multiplicatively to the net, attributed benefit stream.
counterfactual_label	A single character string naming the declared counterfactual. Required when <code>counterfactual = "already_net"</code> ; optional (used only for reporting) when <code>counterfactual</code> is a <code>data.frame</code> .
discount_central	The central real discount rate, as a proportion (CRRDC's Australian rural R&D standard is 0.05).
discount_sensitivity	A numeric vector of sensitivity discount rates reported alongside <code>discount_central</code> (CRRDC's standard band is $c(0.03, 0.07)$).
base_year	The undiscounted period-zero year; defaults to the earliest year across benefits and costs.
grounding	Optional grounding token(s) for the input facts, combined worst-case with <code>combine_grounding()</code> ; defaults to <code>grounding_unverified()</code> when not supplied.

Details

`counterfactual` states what would have happened without the investment. Supply a `data.frame(year, value)` of benefits under the counterfactual scenario, spanning every year benefits covers, and it is subtracted from benefits year by year before attribution and adoption are applied. Alternatively, when benefits already IS the incremental (net-of- counterfactual) stream, pass `counterfactual = "already_net"` and name the declared counterfactual in `counterfactual_label` – the label is required in this branch because a net stream carries no other record of what it is net of.

`attribution` is the proportion, in $(0, 1]$, of the incremental benefit this investment alone caused, net of co-funders, extension effort and other contributing causes. `adoption` is the uptake profile, a `data.frame(year, uptake)` with uptake in $[0, 1]$, applied multiplicatively to the net, attributed benefit stream; it must cover every year benefits covers, so an adoption lag is always stated rather than assumed away. Costs are not adjusted by attribution or adoption – the investment cost is what was actually spent, not a share of a benefit.

The ratio is reported at `discount_central` (the CRRDC standard is 5% for Australian rural R&D) and, side by side, at every rate in `discount_sensitivity` (the CRRDC standard band is 3% and

7%), so a reader sees how sensitive the ratio is to the one assumption a guideline mandates testing. `base_year` (period $t = 0$, undiscounted) defaults to the earliest year across benefits and costs, so an investment whose costs precede its first benefit year is discounted from the year spending began.

Grounding rides the same Independent Oracle Principle firewall as the rest of `decideR`: an ungrounded input (the default when grounding is not supplied) returns an abstained `decider_bcr` with `abstain_reason = "input_ungrounded"` and every rate's `pv_benefits/pv_costs/npv/bcr` NA, never a ratio a funding submission would quote. A discounted cost base that is not strictly positive at the central rate (a zero or negative cost stream) abstains with `abstain_reason = "undefined_ratio"` for the same reason.

Value

A `decider_bcr` object: `rates` (a `data.frame` with one row per discount rate and columns `rate_label`, `rate`, `pv_benefits`, `pv_costs`, `npv`, `bcr`), `counterfactual_label`, `attribution`, `adoption`, `base_year`, `grounding`, `abstained` and `abstain_reason`.

See Also

[decide_bcr\(\)](#) for a benefit-cost ratio without the CRRDC mandatory inputs.

Other value: [decide_bcr\(\)](#), [decide_evpi\(\)](#), [decide_evsi\(\)](#), [valuation\(\)](#)

Examples

```
# A ten-year, evenly-adopted, half-attributed investment. The benefit
# stream is already declared incremental against a named counterfactual.
bens <- data.frame(year = 2020:2029, value = rep(500, 10))
csts <- data.frame(year = 2020:2029, value = rep(1000, 10))
up <- data.frame(year = 2020:2029, uptake = rep(1, 10))
bcr_crrdc(bens, csts, counterfactual = "already_net",
          counterfactual_label = "no new variety released",
          attribution = 0.5, adoption = up,
          grounding = grounding_ungrounded())
```

combine_grounding

Combine input grounding worst-case

Description

A decision that rests on several inputs is grounded only when *every* input is grounded; a single un-grounded input taints the whole decision. This is the worst-case combination mandated by the Independent Oracle Principle, and it is the rule by which a decision inherits its grounding label from the facts it consumes.

Usage

```
combine_grounding(tokens)
```

Arguments

`tokens` A character vector of grounding tokens, each either `grounding_grounded()` or `grounding_unverified()`.

Details

Each token is validated against the canonical vocabulary, so a synonym such as "unverified" is rejected loudly rather than silently mistaken for a grounded fact. The token written by `decideR` 0.2.0 and earlier is accepted as un-grounded.

Value

`grounding_grounded()` if `tokens` is non-empty and every element is grounded; otherwise `grounding_unverified()`.

See Also

[grounding_tokens](#) for the two canonical tokens.

Other grounding: [grounding_tokens](#)

Examples

```
combine_grounding(c(grounding_grounded(), grounding_grounded()))
combine_grounding(c(grounding_grounded(), grounding_unverified()))
```

`decide`

Choose an action by expected utility, with grounding and abstention

Description

The shared decision rule. Given draws from the posterior of a latent state θ , a utility function $U(a, \theta)$, and a set of candidate actions, `decide()` evaluates the expected utility $E_\theta[U(a, \theta)]$ of each feasible candidate by averaging over the draws and selects the maximiser. It declines to do so – abstaining to `safe_action` – under any of four conditions, checked in this order: the combined input grounding is not grounded (the Independent Oracle Principle firewall: never act on an unverified fact); no candidate satisfies constraint; the effective sample size is below `min_ess` (this trigger is opt-in – it never fires unless the caller supplies `min_ess`, since there is no universal floor a decision-agnostic engine can default to); or the chosen action beats the `safe_action` on fewer than `decisive_prob` of the draws (it is not clearly better than the status quo). The last check is judged against the safe action, not the runner-up, so a fine candidate grid does not trigger spurious indecision; when `safe_action` has no numeric match in candidates (a categorical action set, or an off-grid numeric safe action) it falls back to the per-draw argmax probability, so the gate stays live for every candidate shape rather than becoming a no-op.

Usage

```
decide(
  draws,
  utility,
  candidates,
  labels = NULL,
  constraint = NULL,
  grounding = grounding_unverified(),
  safe_action = NULL,
  safe_label = "abstain",
  min_ess = NULL,
  ess = NULL,
  decisive_prob = 0.6,
  method = "expected_utility",
  inputs = list(),
  metadata = list()
)
```

Arguments

<code>draws</code>	A numeric vector of posterior draws of the latent state.
<code>utility</code>	A function function(action, theta) returning a numeric vector of utilities the same length as theta.
<code>candidates</code>	A numeric vector (or list) of candidate actions.
<code>labels</code>	Optional character labels for the candidates; defaults to a formatted candidates.
<code>constraint</code>	Optional function(action) returning TRUE when the action is feasible.
<code>grounding</code>	One or more grounding tokens for the inputs; combined worst-case.
<code>safe_action</code>	The action returned on abstention (e.g. a flat position, a zero input rate, the status quo).
<code>safe_label</code>	A label for the safe action.
<code>min_ess</code>	Optional effective-sample-size floor; the low_ess abstention trigger is off by default and fires only when this is set.
<code>ess</code>	Optional effective sample size of draws; defaults to length(draws).
<code>decisive_prob</code>	The minimum posterior probability that the chosen action beats safe_action, below which the decision abstains as not clearly better than the status quo.
<code>method</code>	A label recording the decision rule.
<code>inputs</code>	A named list recording input provenance.
<code>metadata</code>	A named list of free-form metadata.

Details

The grounding label is combined worst-case across grounding with `combine_grounding()`, so passing the provenance of several inputs taints the decision if any one of them is unverified.

Value

A [decision](#) object.

See Also

[decide_input_rate\(\)](#) and [decide_position_size\(\)](#) for domain wrappers; [decision](#) for the returned object.

Other decisions: [decide_input_rate\(\)](#), [decide_position_size\(\)](#), [decision\(\)](#), [is_grounded\(\)](#)

Examples

```
set.seed(1L)
theta <- rnorm(2000L, mean = 1.2, sd = 0.4)
u <- function(a, theta) -(a - theta)^2
decide(theta, u, candidates = seq(0, 3, by = 0.5),
       grounding = grounding_grounded(), safe_action = 0)
```

decide_bcr

Benefit-cost ratio of a research investment

Description

Discounts a stream of benefits and a stream of costs at an explicit rate over an explicit horizon and reports the components an impact assessment is written from – the two streams, the present value of each, the net present value, and the ratio:

$$PV = \sum_{t=0}^{H-1} \frac{x_t}{(1+r)^t}, \quad NPV = PV_B - PV_C, \quad BCR = PV_B/PV_C.$$

This is the CRRDC-style reporting set an Australian rural research and development impact assessment closes with. `decideR` encodes no guideline default for the rate or the horizon: both are stated by the assessment the analysis serves and are the caller's to supply, so a ratio can never be quoted against an assumption its author did not make.

Usage

```
decide_bcr(
  benefits = NULL,
  costs,
  discount_rate,
  horizon = NULL,
  grounding = NULL,
  manifest = NULL,
  draws_key = "benefit_draws",
  probs = c(0.05, 0.5, 0.95),
  metadata = list()
)
```

Arguments

benefits	The benefit stream: a numeric vector, one value per period from $t = 0$, or a numeric matrix of posterior draws (rows) by periods (columns). May be NULL when manifest supplies it.
costs	The cost stream, a numeric vector, one value per period from $t = 0$.
discount_rate	The real discount rate per period, as a proportion (e.g. 0.05 for five per cent).
horizon	The number of periods, $t = 0, \dots, \text{horizon} - 1$; defaults to the longer of the two streams.
grounding	Optional grounding token(s) for the streams, combined worst-case; defaults to the manifest's own token, or to <code>grounding_unverified()</code> when no manifest is supplied.
manifest	Optional <code>orchestra_manifest</code> S7 object supplying the benefit stream and its grounding.
draws_key	The metadata key under which the benefit draws are sought when the manifest's outputs slot is empty.
probs	Quantiles at which the posterior ratio is summarised when the benefit stream arrives as draws.
metadata	A named list of free-form metadata.

Details

Period $t = 0$ is the present and is not discounted. A stream given as end-of-period cash flows is expressed by leading it with a zero, as in the worked example below. Streams shorter than the horizon are zero-padded; a stream longer than the horizon is an error rather than a silent truncation.

When manifest is supplied, the benefit stream and its grounding token are read from it through the same duck-typed contract accessors the decision tail uses (`outputs`, then `metadata[[draws_key]]`; `grounding` from `metadata$grounding` or `summary$metrics$grounding`), with draws in rows and periods in columns. The point estimate then uses the posterior mean stream, and the posterior distribution of the ratio is summarised in `components$bcr_quantiles`. An ungrounded manifest returns an abstained valuation, never a ratio.

Value

A [valuation](#) of type = "bcr" whose value is the ratio and whose components carry the discounted streams, the two present values and the net present value.

See Also

[decide_evsi\(\)](#) for the value of the next study rather than of the whole programme.

Other value: [bcr_crrdc\(\)](#), [decide_evpi\(\)](#), [decide_evsi\(\)](#), [valuation\(\)](#)

Examples

```
# A two-period investment: 100 spent now, 210 returned next period, at a
# five per cent discount rate. PV(costs) = 100, PV(benefits) = 210/1.05 =
# 200, so NPV = 100 and BCR = 2.
```

```

v <- decide_bcr(benefits = c(0, 210), costs = c(100, 0),
               discount_rate = 0.05,
               grounding = grounding_grounded())
v@value
v@components$npv

# The same investment on an un-grounded benefit stream prices out as an
# abstention rather than as a ratio.
decide_bcr(benefits = c(0, 210), costs = c(100, 0), discount_rate = 0.05)

```

decide_evpi

Expected value of perfect information from a manifest

Description

Prices what it would be worth to learn the latent state exactly before acting. Given the same manifest, candidate set and utility that `decide_from_manifest()` takes, `decide_evpi()` returns

$$EVPI = E_{\theta}[\max_a U(a, \theta)] - \max_a E_{\theta}[U(a, \theta)],$$

the difference between deciding with the state known and deciding with only its posterior. It is the ceiling on the value of any study, trial or measurement addressing that state: no finite sample can be worth more, which is what makes it the natural first screen before an experiment is designed.

Usage

```

decide_evpi(
  manifest,
  candidates,
  utility = NULL,
  grounding = NULL,
  safe_action = NULL,
  draws_key = "yield_draws",
  ...
)

```

Arguments

manifest	An orchestra_manifest S7 object (or any S7 object exposing the contract's outputs / metadata / summary properties).
candidates	A numeric vector (or list) of candidate actions.
utility	A function function(action, draws_column) returning a numeric utility per draw, or NULL when each manifest column is already the utility of the corresponding candidate.
grounding	Optional override for the manifest's own grounding token.
safe_action	The action the baseline decision abstains to; defaults to the first candidate.

draws_key	The metadata key under which a draws matrix is sought when the outputs slot is empty.
...	Further arguments passed to the decision core (e.g. constraint, min_ess, ess, decisive_prob).

Details

The expectation runs over the feasible candidates only, so a hard constraint passed through ... prices the information available *within* the actions the grower or the agency can actually take, not within a wider hypothetical set.

Grounding rides from the producer exactly as it does for a decision: an ungrounded manifest returns an abstained valuation with `abstain_reason = "input_ungrounded"` and `value = NA_real_`. The other blocking reasons are `no_feasible_action`, `low_ess` and `degenerate_utility`. An insufficient_evidence baseline decision is *not* blocking: a too-close-to-call decision is precisely where information pays, and the indecisive decision is returned in `@baseline`.

Value

A [valuation](#) of type = "evpi". Its components carry the two expectations the value is the difference of, the draw count and the number of feasible candidates; `@baseline` carries the decision current evidence supports.

References

Raiffa, H. and Schlaifer, R. (1961) *Applied Statistical Decision Theory*. Boston: Harvard University Graduate School of Business Administration.

See Also

[decide_evsi\(\)](#) for the value of one finite sample; [decide_from_manifest\(\)](#) for the decision the value is measured against.

Other value: [bcr_crrdc\(\)](#), [decide_bcr\(\)](#), [decide_evsi\(\)](#), [valuation\(\)](#)

Examples

```
set.seed(1L)
rates <- seq(0, 200, by = 25)
demo_manifest <- S7::new_class(
  "demo_manifest",
  properties = list(
    outputs      = S7::new_property(S7::class_any, default = NULL),
    metadata     = S7::new_property(S7::class_list, default = list()),
    summary      = S7::new_property(S7::class_any, default = NULL),
    run_id       = S7::new_property(S7::class_character, default = ""),
    emitter_package = S7::new_property(S7::class_character, default = "")
  )
)
ymax <- rnorm(800L, 5.5, 0.6)
yld <- vapply(rates,
  function(r) 3 + (ymax - 3) * (1 - exp(-r / 80)),
```

```

      numeric(800L))
m <- demo_manifest(
  metadata = list(yield_draws = yld, grounding = grounding_grounded())
)
decide_evpi(m, candidates = rates,
  utility = function(r, y) 300 * y - 1.2 * r)

```

decide_evsi

Expected value of one prospective sample from a manifest

Description

Prices a named, finite study before it is run: this soil test, this trial, at this site. Where `decide_evpi()` prices perfect knowledge, `decide_evsi()` prices the imperfect knowledge a real sample would deliver,

$$EVSI = E_y[\max_a E_{\theta|y}[U(a, \theta)]] - \max_a E_{\theta}[U(a, \theta)],$$

the expected gain from being allowed to decide after seeing the data rather than before. It is bounded above by the EVPI, which the valuation reports beside it as a ceiling.

Usage

```

decide_evsi(
  manifest,
  candidates,
  sample_design,
  utility = NULL,
  state = NULL,
  state_key = NULL,
  n_pre = 500L,
  grounding = NULL,
  safe_action = NULL,
  draws_key = "yield_draws",
  ...
)

```

Arguments

<code>manifest</code>	An <code>orchestra_manifest</code> S7 object, as for <code>decide_evpi()</code> .
<code>candidates</code>	A numeric vector (or list) of candidate actions.
<code>sample_design</code>	A named list describing the prospective sample: <code>simulate</code> (a function(<code>state</code>) returning one hypothetical dataset drawn at that latent state), <code>loglik</code> (a function(<code>y</code> , <code>state</code>) returning the log-likelihood of that dataset at every state draw, one value per draw), and optionally <code>label</code> , <code>n_obs</code> and <code>cost</code> .
<code>utility</code>	A function function(<code>action</code> , <code>draws_column</code>), or <code>NULL</code> when each manifest column is already the utility of its candidate.

state	Optional numeric vector, one latent-state value per draw, that the prospective sample informs.
state_key	Optional name of a column of the manifest's params data.frame to read the state from.
n_pre	The number of outer (pre-posterior) draws.
grounding	Optional override for the manifest's own grounding token.
safe_action	The action the baseline decision abstains to.
draws_key	The metadata key under which a draws matrix is sought when the outputs slot is empty.
...	Further arguments passed to the decision core (e.g. constraint, decisive_prob).

Details

The pre-posterior expectation is evaluated by the standard two-loop Monte Carlo algorithm: draw a latent state from the current posterior, simulate the dataset the sample would produce at that state, update, and re-decide. The inner update here is an importance re-weighting of the manifest's own draws by the sampling log-likelihood rather than a nested re-fit, so the manifest stays the single source of the posterior and decideR adds no sampler. That re-weighting degenerates when the prospective sample is very much more informative than the current posterior; the effective sample size of the weights is therefore computed at every outer draw, reported in `components$weight_ess_min`, and warned about when it collapses.

The estimate is a Monte Carlo one and is reported unclamped. A small negative value is Monte Carlo noise around a true value near zero – a sample worth nothing – and is not silently floored at zero.

Value

A [valuation](#) of type = "evsi", whose components carry the pre-posterior and current-information expectations, the EVPI ceiling, the weight diagnostics, and – when `sample_design$cost` is supplied – the cost and the net value.

References

- Raiffa, H. and Schlaifer, R. (1961) *Applied Statistical Decision Theory*. Boston: Harvard University Graduate School of Business Administration.
- Ades, A. E., Lu, G. and Claxton, K. (2004) Expected value of sample information calculations in medical decision modeling. *Medical Decision Making*, 24(2), 207–227. doi:[10.1177/0272989X04263162](https://doi.org/10.1177/0272989X04263162)

See Also

[decide_evpi\(\)](#) for the ceiling on any sample's value.

Other value: [bcr_crrdc\(\)](#), [decide_bcr\(\)](#), [decide_evpi\(\)](#), [valuation\(\)](#)

Examples

```
set.seed(2L)
demo_manifest <- S7::new_class(
  "demo_manifest",
  properties = list(
    outputs      = S7::new_property(S7::class_any, default = NULL),
    metadata     = S7::new_property(S7::class_list, default = list()),
    summary      = S7::new_property(S7::class_any, default = NULL),
    run_id       = S7::new_property(S7::class_character, default = ""),
    emitter_package = S7::new_property(S7::class_character, default = "")
  )
)
# A posterior for the yield gain from a nitrogen top-up, in t/ha.
gain <- rnorm(600L, mean = 0.10, sd = 0.35)
m <- demo_manifest(
  outputs = matrix(gain, ncol = 1L),
  metadata = list(grounding = grounding_grounded())
)
# A four-plot strip trial measuring the gain, observation sd 0.5.
trial <- list(
  label    = "4-plot strip trial",
  n_obs    = 4L,
  cost     = 0.02,
  simulate = function(state) mean(rnorm(4L, state, 0.5)),
  loglik   = function(y, state) dnorm(y, state, 0.5 / sqrt(4), log = TRUE)
)
decide_evsi(m, candidates = c(0, 1), sample_design = trial,
  utility = function(a, g) a * g, n_pre = 200L)
```

decide_from_manifest *Turn an orchestra manifest into a decision*

Description

The manifest-native pipeline tail. Given an orchestra_manifest emitted by any ORCHESTRA member – a PESTO ensemble inversion, a kernR causal test, a flexyBayes posterior – decide_from_manifest() extracts the predictive draws and the producer’s grounding token from the manifest, then routes them through the same expected-utility engine [decide\(\)](#) uses. It is the standard closer of the crop pipeline: an upstream member’s inference becomes a loss-optimal, grounding-aware action without the caller re-assembling the draws by hand.

Usage

```
decide_from_manifest(
  manifest,
  candidates,
  utility = NULL,
  grounding = NULL,
  safe_action = NULL,
```

```

    draws_key = "yield_draws",
    ...
)

```

Arguments

manifest	An orchestra_manifest S7 object (or any S7 object exposing the contract's outputs / metadata / summary properties).
candidates	A numeric vector of candidate actions, one per column of the manifest's draws matrix; or a numeric vector the utility indexes freely when the draws are a single posterior vector.
utility	A function function(action, draws_column) returning a numeric utility per draw, or NULL to treat each manifest column as the utility of the corresponding candidate directly (the case in which a producer has already evaluated the loss per candidate).
grounding	Optional override for the manifest's own grounding token; defaults to the token the manifest carries, read worst-case.
safe_action	The action returned on abstention; defaults to the first candidate (the status-quo convention for an ordered grid).
draws_key	The metadata key under which a draws matrix is sought when the outputs slot is empty.
...	Further arguments passed to the decision core (e.g. min_ess, ess, decisive_prob, safe_label, method, metadata).

Details

The grounding label rides from the producer through the decision: a manifest whose payload is ungrounded forces an abstention to `safe_action`, regardless of how attractive the draws look. This is the Independent Oracle Principle firewall applied at the pipeline boundary – `decideR` never mints a confident recommendation on an upstream fact the producer did not ground.

`decideR` does not depend on the contract package; the manifest is read through its public S7 properties (outputs, metadata, summary). The draws are sought in the outputs slot first, then under `metadata[[draws_key]]`, so the tail consumes any conforming producer rather than special-casing each one.

Value

A [decision](#) object whose inputs record the manifest's `run_id` and `emitter_package` for provenance.

See Also

[decide_rate_from_manifest\(\)](#) for the agronomic convenience; [decide\(\)](#) for the underlying engine.

Other manifest: [decide_rate_from_manifest\(\)](#)

Examples

```

set.seed(1L)
rates <- seq(0, 200, by = 25)
# In production the manifest is a real `orchestra_manifest` S7 object from
# the composition layer. Here, a minimal S7 stand-in carrying the same
# public properties the tail reads (`outputs`, `metadata`, `summary`,
# `run_id`, `emitter_package`) -- an S3 list is NOT readable by the tail,
# which duck-types through S7 property access.
demo_manifest <- S7::new_class(
  "demo_manifest",
  properties = list(
    outputs      = S7::new_property(S7::class_any, default = NULL),
    metadata     = S7::new_property(S7::class_list, default = list()),
    summary      = S7::new_property(S7::class_any, default = NULL),
    run_id       = S7::new_property(S7::class_character, default = ""),
    emitter_package = S7::new_property(S7::class_character, default = "")
  )
)
ymax <- rnorm(1500L, 5.5, 0.3)
yld <- vapply(rates,
  function(r) 3 + (ymax - 3) * (1 - exp(-r / 80)),
  numeric(1500L))
m <- demo_manifest(
  metadata = list(yield_draws = yld, grounding = grounding_grounded())
)
decide_from_manifest(m, candidates = rates,
  utility = function(r, y) 300 * y - 1.2 * r)

```

decide_input_rate	<i>Choose a crop input rate by expected profit</i>
-------------------	--

Description

A reference decision for the agronomic domain. Given posterior draws of yield at each candidate input rate – the kind of object a mechanistic-simulator inversion (PESTO) or a causal test (kernR) emits – it selects the rate that maximises expected profit, $E[p_y y(r) - p_r r]$, where p_y is the grain price and p_r the input price. It abstains to `safe_rate` when the yield evidence is un-grounded or the leading rate is not decisive.

Usage

```

decide_input_rate(
  yield_draws,
  rates,
  price_grain,
  price_input,
  grounding = grounding_unverified(),
  safe_rate = 0,
  decisive_prob = 0.6,

```

```
    ...
  )
```

Arguments

yield_draws	A numeric matrix of posterior yield draws, one column per candidate rate (rows are draws).
rates	A numeric vector of candidate input rates, one per column of yield_draws.
price_grain	Grain price per unit yield.
price_input	Input price per unit rate.
grounding	One or more grounding tokens for the yield evidence; combined worst-case.
safe_rate	The fallback rate on abstention (default 0, the no-input status quo).
decisive_prob	Minimum posterior probability that the leading rate is best (see decide()).
...	Further arguments passed to the decision core: min_ess, ess, constraint (a function(rate) feasibility check), safe_label, method, metadata, inputs. Any other name stops with an error rather than being silently discarded.

Value

A [decision](#) object whose action is the chosen rate.

See Also

[decide\(\)](#) for the underlying engine; [decide_position_size\(\)](#) for the trading counterpart.

Other decisions: [decide\(\)](#), [decide_position_size\(\)](#), [decision\(\)](#), [is_grounded\(\)](#)

Examples

```
set.seed(1L)
rates <- seq(0, 200, by = 40)
# diminishing-returns yield response with posterior noise
mu <- 3 + 2.5 * (1 - exp(-rates / 80))
yield_draws <- sapply(mu, function(m) rnorm(1500L, m, 0.3))
decide_input_rate(yield_draws, rates, price_grain = 300, price_input = 1.2,
                  grounding = grounding_grounded())
```

decide_position_size	<i>Size a trading position by capped expected log-growth under a loss-quantile limit</i>
----------------------	--

Description

A reference decision for the trading domain. Given predictive draws of the next-period simple return, it chooses the fraction of capital that maximises expected log-growth $E[\log(1 + f r)]$ (the Kelly objective; Kelly, 1956) over a grid bounded by `kelly_cap`, subject to a hard one-period loss-quantile constraint: a fraction is feasible only if its `loss_quantile` return quantile is no worse than `-max_drawdown`. **This guard is a single-period value-at-risk floor, not a path drawdown** – `-max_drawdown` names the worst `loss_quantile`-fraction of *next-period* returns the position may be exposed to, not a peak-to-trough statistic accumulated over a trading path; a true multi-period drawdown limit is not computed here. With no edge the grid maximiser is naturally flat; with an edge the constraint caps the size. It abstains to flat (0) when the return evidence is un-grounded – the capital firewall: never stake real money on a market fact that has not been verified against the exchange.

Usage

```
decide_position_size(
    return_draws,
    capital,
    max_drawdown,
    kelly_cap = 0.5,
    n_grid = 41L,
    loss_quantile = 0.05,
    grounding = grounding_unverified(),
    decisive_prob = 0.6,
    ...
)
```

Arguments

<code>return_draws</code>	A numeric vector of predictive next-period simple returns.
<code>capital</code>	The capital available; the dollar position is <code>fraction * capital</code> .
<code>max_drawdown</code>	The hard one-period loss-quantile cap (e.g. 0.15) – a value-at-risk floor on the <i>next-period</i> return, not a multi-period peak-to-trough drawdown.
<code>kelly_cap</code>	The maximum fraction of capital (a fractional-Kelly ceiling).
<code>n_grid</code>	The number of fractions evaluated on <code>[0, kelly_cap]</code> .
<code>loss_quantile</code>	The lower return quantile the <code>max_drawdown</code> guard is evaluated at.
<code>grounding</code>	One or more grounding tokens for the return evidence; combined worst-case.
<code>decisive_prob</code>	Minimum posterior probability that the leading fraction is best (see decide()).
<code>...</code>	Further arguments passed to the decision core: <code>min_ess</code> , <code>ess</code> , <code>constraint</code> (a function(<code>fraction</code>) feasibility check, ANDed onto the hard drawdown constraint – it can only tighten feasibility, never loosen it), <code>safe_label</code> , <code>method</code> , <code>metadata</code> , <code>inputs</code> . Any other name stops with an error rather than being silently discarded.

Value

A [decision](#) object whose action is the chosen fraction of capital; the implied dollar position is in `@metadata$position`.

References

Kelly, J. L. (1956). A New Interpretation of Information Rate. *Bell System Technical Journal*, 35(4), 917-926.

See Also

[decide\(\)](#) for the underlying engine; [decide_input_rate\(\)](#) for the agronomic counterpart.

Other decisions: [decide\(\)](#), [decide_input_rate\(\)](#), [decision\(\)](#), [is_grounded\(\)](#)

Examples

```
set.seed(1L)
# a small positive edge with fat tails
r <- 0.004 + 0.02 * rt(4000L, df = 4) / sqrt(2)
decide_position_size(r, capital = 1000, max_drawdown = 0.15,
                    grounding = grounding_grounded())
```

decide_rate_from_manifest

Choose a crop input rate from an upstream yield manifest

Description

The agronomic manifest tail. Where [decide_input_rate\(\)](#) takes a yield-draws matrix assembled by the caller, `decide_rate_from_manifest()` sources that matrix – and the grounding token – directly from an `orchestra_manifest` emitted upstream (a PESTO inversion or a mechanistic yield-response posterior). It is the formalised crop-pipeline tail: a yield posterior in, a profit-optimal, grounding-aware nitrogen-rate decision out. The grounding rides from the producer, so an un-grounded yield posterior forces the status-quo rate.

Usage

```
decide_rate_from_manifest(
  manifest,
  rates,
  price_grain,
  price_input,
  safe_rate = 0,
  decisive_prob = 0.6,
  ...
)
```

Arguments

manifest	An orchestra_manifest carrying a yield-per-rate draws matrix (in outputs or metadata\$yield_draws), one column per rate.
rates	A numeric vector of candidate input rates, one per column of the manifest's draws matrix.
price_grain	Grain price per unit yield.
price_input	Input price per unit rate.
safe_rate	The fallback rate on abstention (default 0, the no-input status quo).
decisive_prob	Minimum posterior probability that the leading rate is best (see decide()).
...	Further arguments passed to the decision core (e.g. min_ess).

Value

A [decision](#) object whose action is the chosen rate.

See Also

[decide_from_manifest\(\)](#) for the generic tail; [decide_input_rate\(\)](#) for the caller-assembled-matrix form.

Other manifest: [decide_from_manifest\(\)](#)

Examples

```
set.seed(1L)
rates <- seq(0, 200, by = 25)
demo_manifest <- S7::new_class(
  "demo_manifest",
  properties = list(
    outputs      = S7::new_property(S7::class_any, default = NULL),
    metadata     = S7::new_property(S7::class_list, default = list()),
    summary      = S7::new_property(S7::class_any, default = NULL),
    run_id       = S7::new_property(S7::class_character, default = ""),
    emitter_package = S7::new_property(S7::class_character, default = "")
  )
)
ymax <- rnorm(1500L, 5.5, 0.3)
yld  <- vapply(rates,
  function(r) 3 + (ymax - 3) * (1 - exp(-r / 80)),
  numeric(1500L))
m <- demo_manifest(
  metadata = list(yield_draws = yld, grounding = grounding_grounded())
)
decide_rate_from_manifest(m, rates, price_grain = 300, price_input = 1.2)
```

decision	<i>A decision-theoretic action with grounding and abstention</i>
----------	--

Description

The object returned by `decide()` and the reference decisions. It records the chosen action, the `expected_utility` that selected it, the complete candidates ledger (every action with its expected utility and feasibility), and the two provenance fields the Independent Oracle Principle requires: `grounding` (`grounding_grounded()` or `grounding_unverified()`) and the abstention record (`abstained`, `abstain_reason`). When a decision abstains the action is the caller's `safe_action`, not the utility-maximal one.

Usage

```
decision(
  action = NULL,
  action_label = NA_character_,
  expected_utility = NA_real_,
  candidates = NULL,
  grounding = "ungrounded",
  abstained = FALSE,
  abstain_reason = NA_character_,
  safe_action = NULL,
  method = NA_character_,
  inputs = list(),
  metadata = list()
)
```

Arguments

<code>action</code>	The chosen action. The <code>safe_action</code> when <code>abstained</code> is TRUE.
<code>action_label</code>	A short human-readable label for the action; suffixed "ungrounded" when the grounding is not grounded.
<code>expected_utility</code>	The expected (mean-over-draws) utility of the chosen action, or <code>NA_real_</code> when abstained.
<code>candidates</code>	A <code>data.frame</code> with one row per candidate action: <code>action</code> , <code>expected_utility</code> , <code>feasible</code> .
<code>grounding</code>	A canonical grounding token.
<code>abstained</code>	TRUE when the decision declined the utility-maximal action.
<code>abstain_reason</code>	A short reason string, or <code>NA_character_</code> .
<code>safe_action</code>	The fallback action taken on abstention.
<code>method</code>	The decision rule applied (e.g. "expected_profit").
<code>inputs</code>	A named list recording input provenance.
<code>metadata</code>	A named list of free-form metadata.

Value

A decision S7 object.

See Also

[decide\(\)](#) and the reference decisions that produce this object.

Other decisions: [decide\(\)](#), [decide_input_rate\(\)](#), [decide_position_size\(\)](#), [is_grounding\(\)](#)

Examples

```
d <- decision(action = 120, action_label = "rate=120 kg/ha",
              expected_utility = 1840, grounding = grounding_grounded())
d@action
```

grade_band_utility	<i>Build a decide() utility from a grade-band schedule</i>
--------------------	--

Description

Turns a [grade_band_value](#) schedule into a utility(action, theta) function the decision engine consumes. theta is a posterior draw of the quality outcome (grain protein, say); the action shifts that outcome through quality_shift, a function function(action, theta) returning the realised quality under the action. The utility of an action at a draw is the band payoff of the shifted outcome, less an optional per-unit action cost. Because the payoff is a step function, the expected utility decideR averages over the draws is governed by each band's posterior probability – exactly the threshold economics a quality-graded decision faces.

Usage

```
grade_band_utility(value, quality_shift = NULL, cost = 0)
```

Arguments

value	A grade_band_value schedule.
quality_shift	A function function(action, theta) returning the realised quality outcome under the action, or NULL for the additive default theta + action.
cost	A function function(action) returning the action's cost, or a single numeric per-unit cost applied as cost * action, or 0 for no cost.

Details

When quality_shift is left NULL the action is taken to *be* the quality outcome offset added to theta (the common case where an input nudges the outcome additively), so quality = theta + action. Supply a quality_shift for any non-additive response (a saturating dose-response, say).

Value

A function function(action, theta) suitable as the utility argument to `decide()`.

See Also

`grade_band_value()` for the schedule; `decide()` for the engine.
Other loss: `grade_band_value()`

Examples

```
v <- grade_band_value(c(10.5, 11.5, 13), c(300, 330, 360, 400))
# A nitrogen rate lifts protein additively; protein costs nothing but N does.
u <- grade_band_utility(v, quality_shift = function(rate, protein)
  protein + 0.01 * rate, cost = 1.2)

set.seed(1L)
protein <- rnorm(2000L, mean = 11.0, sd = 0.6)
decide(protein, u, candidates = seq(0, 150, by = 25),
  grounding = grounding_grounded(), safe_action = 0)
```

grade_band_value	<i>A grade-band value schedule (threshold-economics loss)</i>
------------------	---

Description

Builds a step value schedule: a payoff that is constant within a quality band and jumps at each breakpoint. This is the canonical shape of a quality-graded payoff – a grain price that steps up as protein clears 10.5%, 11.5%, and 13%, say – where the value of an outcome depends on which band it falls in, not on its exact magnitude. The returned object is a vectorised function from a quality outcome to its band payoff, classed so it carries the schedule for inspection and prints as a readable band table.

Usage

```
grade_band_value(breaks, values, right = FALSE, labels = NULL)
```

Arguments

breaks	A strictly increasing numeric vector of band edges (e.g. the protein thresholds <code>c(10.5, 11.5, 13)</code>).
values	A numeric vector of band payoffs, of length <code>length(breaks) + 1</code> , ordered from the lowest band to the highest.
right	Logical; when FALSE (default) bands are <code>[lower, upper)</code> so an outcome on a break clears into the higher band; when TRUE bands are <code>(lower, upper]</code> . [lower,upper)so an outcome on a break clears into the higher band; whenTRUEbands are(lower,upper]: R:lower,%20upper)%60%20so%20an%0Aoutcome%20on%20a%20break%20clears%20into%20
labels	Optional character labels for the bands, of length <code>length(values)</code> ; defaults to formatted interval strings.

Details

breaks are the band edges, strictly increasing. values are the payoffs, one per band: the first applies below breaks[1], the last applies at or above breaks[length(breaks)], and each interior element applies between a consecutive pair of breaks. With right = FALSE (the default) each band is the half-open interval [lower, upper), so an outcome exactly on a break clears into the higher band – the convention a delivery standard uses when it pays the better grade at the threshold.

Value

A function of one numeric argument returning the band payoff for each element, with class grade_band_value and the schedule stored as attributes (breaks, values, labels, right).

See Also

[grade_band_utility\(\)](#) to turn the schedule into a decide() utility; [decide\(\)](#) for the engine that consumes it.

Other loss: [grade_band_utility\(\)](#)

Examples

```
# An APW-style protein payoff: $/t steps up as protein clears each band.
v <- grade_band_value(breaks = c(10.5, 11.5, 13),
                     values = c(300, 330, 360, 400))
v(c(9.0, 11.0, 11.5, 14.0))
v
```

grounding_tokens

Canonical grounding tokens

Description

The two states a fact – or a decision assembled from facts – can occupy under the Independent Oracle Principle. grounding_grounded() returns the token for a fact that has been checked against its external authority; grounding_unverified() returns the token for everything else, which is the default until a grounding check proves otherwise.

Usage

```
grounding_grounded()
```

```
grounding_unverified()
```

Details

grounding_unverified() returns "ungrounded". The token written by decideR 0.2.0 and earlier is still accepted wherever a token is read.

Value

A length-one character string holding the canonical token.

See Also

[combine_grounding\(\)](#) for worst-case combination across inputs.

Other grounding: [combine_grounding\(\)](#)

Examples

```
grounding_grounded()
grounding_unverified()
```

is_grounded	<i>Is a decision grounded?</i>
-------------	--------------------------------

Description

A convenience predicate reading the decision's grounding label. It reports only the grounding of the evidence, not whether the decision abstained – `is_grounded()` is TRUE for an abstained decision whenever the abstention reason was `no_feasible_action`, `low_ess`, or `insufficient_evidence`, since none of those reasons downgrades the grounding label itself. A consumer that must refuse to act on either un-grounded evidence or an abstained decision – for example a live trading order path – gates on `is_grounded(x) && !x@abstained`, not on `is_grounded()` alone.

Usage

```
is_grounded(x)
```

Arguments

x A decision.

Value

TRUE when the decision's grounding is `grounding_grounded()`. This does not imply the decision was not abstained; see Details.

See Also

Other decisions: [decide\(\)](#), [decide_input_rate\(\)](#), [decide_position_size\(\)](#), [decision\(\)](#)

Examples

```
is_grounded(decision(grounding = grounding_grounded()))
is_grounded(decision(grounding = grounding_unverified()))
```

```
print,decideR::decision-method
      Print a decision
```

Description

Print a decision

Arguments

`x` A [decision](#).
`...` Ignored, for compatibility with [print\(\)](#).

Value

`x`, invisibly.

Examples

```
print(decision(action_label = "flat", grounding = grounding_grounded()))
```

```
print,decideR::valuation-method
      Print a valuation
```

Description

Print a valuation

Arguments

`x` A [valuation](#).
`...` Ignored, for compatibility with [print\(\)](#).

Value

`x`, invisibly.

Examples

```
print(valuation(value = 12.4, type = "evpi",
                grounding = grounding_grounded()))
```

print.decider_bcr	<i>Print a CRRDC benefit-cost ratio</i>
-------------------	---

Description

Print a CRRDC benefit-cost ratio

Usage

```
## S3 method for class 'decider_bcr'  
print(x, ...)
```

Arguments

x	A <code>bcr_crrdc()</code> result.
...	Ignored, for compatibility with <code>print()</code> .

Value

x, invisibly.

Examples

```
bens <- data.frame(year = 2020:2021, value = c(0, 210))  
csts <- data.frame(year = 2020:2021, value = c(100, 0))  
up <- data.frame(year = 2020:2021, uptake = c(1, 1))  
print(bcr_crrdc(bens, csts, counterfactual = "already_net",  
               counterfactual_label = "no intervention", attribution = 1,  
               adoption = up, grounding = grounding_grounded()))
```

print.grade_band_value	<i>Print a grade-band value schedule</i>
------------------------	--

Description

Print a grade-band value schedule

Usage

```
## S3 method for class 'grade_band_value'  
print(x, ...)
```

Arguments

x	A <code>grade_band_value</code> schedule.
...	Ignored, for compatibility with <code>print()</code> .

Value

x, invisibly.

Examples

```
print(grade_band_value(c(10.5, 11.5), c(300, 330, 360)))
```

valuation	<i>A priced quantity with grounding and abstention</i>
-----------	--

Description

The object returned by `decide_evpi()`, `decide_evsi()` and `decide_bcr()`. It records the priced value, the type of valuation that produced it, the arithmetic components a reader needs in order to audit that value, and the two Independent Oracle Principle provenance fields: grounding, and the abstention record (abstained, abstain_reason). When a valuation abstains, value is NA_real_ – decideR never mints a dollar figure on evidence its producer did not ground.

Usage

```
valuation(  
  value = NA_real_,  
  type = NA_character_,  
  components = list(),  
  grounding = "ungrounded",  
  abstained = FALSE,  
  abstain_reason = NA_character_,  
  baseline = NULL,  
  method = NA_character_,  
  inputs = list(),  
  metadata = list()  
)
```

Arguments

value	The priced quantity, in the utility units of the inputs (currency per unit area for a crop valuation, a dimensionless ratio for a benefit-cost ratio), or NA_real_ when abstained.
type	A short valuation type: "evpi", "evsi" or "bcr".
components	A named list of the parts the value is assembled from, so the number can be audited rather than trusted.
grounding	A canonical grounding token.
abstained	TRUE when the valuation declined to price the quantity.
abstain_reason	A short reason string, or NA_character_.

baseline	The decision the current evidence supports, for a value of information; NULL otherwise.
method	A label recording the valuation rule.
inputs	A named list recording input provenance.
metadata	A named list of free-form metadata.

Details

The `abstain_reason` vocabulary extends the decision vocabulary with the reasons specific to pricing: `input_ungrounded`, `no_feasible_action`, `low_ess`, `degenerate_utility` (all inherited from the decision that the valuation is built on) and `undefined_ratio` (a benefit-cost ratio whose discounted cost base is not strictly positive).

Value

A valuation S7 object.

See Also

[decide_evpi\(\)](#), [decide_evsi\(\)](#), [decide_bcr\(\)](#).

Other value: [bcr_crrdc\(\)](#), [decide_bcr\(\)](#), [decide_evpi\(\)](#), [decide_evsi\(\)](#)

Examples

```
v <- valuation(value = 12.4, type = "evpi",
               grounding = grounding_grounded())
v@value
```

Index

- * **decisions**
 - decide, 6
 - decide_input_rate, 16
 - decide_position_size, 17
 - decision, 21
 - is_grounded, 25
 - print, decideR::decision-method, 26
- * **grounding**
 - combine_grounding, 5
 - grounding_tokens, 24
- * **loss**
 - grade_band_utility, 22
 - grade_band_value, 23
- * **manifest**
 - decide_from_manifest, 14
 - decide_rate_from_manifest, 19
- * **value**
 - bcr_crrdc, 3
 - decide_bcr, 8
 - decide_evpi, 10
 - decide_evsi, 12
 - print, decideR::valuation-method, 26
 - valuation, 28

as.data.frame(), 3

as.data.frame.decider_bcr, 2

bcr_crrdc, 3

bcr_crrdc(), 3, 9, 11, 13, 27, 29

combine_grounding, 5

combine_grounding(), 4, 7, 25

decide, 6

decide(), 14, 15, 17–25

decide_bcr, 8

decide_bcr(), 3, 5, 11, 13, 28, 29

decide_evpi, 10

decide_evpi(), 5, 9, 12, 13, 28, 29

decide_evsi, 12

decide_evsi(), 5, 9, 11, 28, 29

decide_from_manifest, 14

decide_from_manifest(), 10, 11, 20

decide_input_rate, 16

decide_input_rate(), 8, 19, 20, 22, 25

decide_position_size, 17

decide_position_size(), 8, 17, 22, 25

decide_rate_from_manifest, 19

decide_rate_from_manifest(), 15

decision, 8, 15, 17, 19, 20, 21, 26

decision(), 8, 17, 19, 25

grade_band_utility, 22

grade_band_utility(), 24

grade_band_value, 22, 23, 27

grade_band_value(), 23

grounding_grounded(grounding_tokens), 24

grounding_tokens, 6, 24

grounding_unverified(grounding_tokens), 24

grounding_unverified(), 4

is_grounded, 25

is_grounded(), 8, 17, 19, 22

print(), 26, 27

print, decideR::decision-method, 26

print, decideR::valuation-method, 26

print.decider_bcr, 27

print.grade_band_value, 27

valuation, 9, 11, 13, 26, 28

valuation(), 5, 9, 11, 13