# Package: msemodules (via r-universe)

June 4, 2026

**Title** Modules and Functions for the mse Package

**Version** 0.2.1.9003

**Description** A number of additional modules and auxiliary functions for the 'mse' package are provided here.

**URL** https://msemodules.flrproject.org/, https://github.com/flr/msemodules

**BugReports** https://github.com/flr/msemodules/issues

**Additional_repositories** https://flr.r-universe.dev

**Depends** R(>= 4.1), FLCore, mse

**Imports** methods, ggplot2, ggplotFL, FLasher, data.table

**License** EUPL-1.2

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Suggests** testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Repository** https://iagomosqueira.r-universe.dev

**Date/Publication** 2026-05-25 17:26:54 UTC

**RemoteUrl** https://github.com/flr/msemodules

**RemoteRef** devel

**RemoteSha** a700b5a943dc8032c99564f55afad6679e7318fd

## Contents

---

`bank_borrow.is` *Banking and borrowing implementation system method*

---

## Description

Adjusts a TAC recommendation from an HCR by banking a proportion of any increase or borrowing against a future quota when a decrease is large, smoothing inter-annual TAC variability while respecting stock health conditions.

## Usage

```
bank_borrow.is(
  stk,
  ctrl,
  args,
  split = NULL,
  rate = NULL,
  diff = 0.15,
  healthy = 1,
  tracking
)
```

## Arguments

| | |
|---|---|
| `stk` | An FLCore::FLStock object representing the stock. |
| `ctrl` | An FLasher::fwdControl object carrying the TAC recommendation to be adjusted. |
| `args` | A list of management-cycle dimensionality arguments supplied automatically by `mse::mp()`. Must contain at least `iy`, `ay`, `frq`, and `management_lag`. |
| `split` | Optional split-control input. Currently unused; reserved for future support of multiple mse::mseCtrl objects within a single `isys` step. Defaults to `NULL`. |
| `rate` | Numeric in $[0, 1]$. Proportion of the TAC to bank or borrow when the threshold condition is met. Must be supplied; there is no default. |
| `diff` | Numeric. Minimum relative change in TAC (compared to the previous HCR decision `pre`) required to trigger a banking or borrowing action. Defaults to `0.15`, i.e. a 15% change. |

`healthy` Numeric. Minimum value of the `rule.hcr` tracking metric required for the stock to be considered healthy enough for banking or borrowing to be applied. Defaults to 1.

`tracking` An FLCore::FLQuant used to record intermediate values and decisions during MP evaluation, passed through and updated by `mse::mp()`.

## Details

In the initial year (`ay == iy`) the tracking metrics `borrowing.isys` and `banking.isys` are initialized to zero and the reference TAC (`pre`) is set to the realized catch in the year preceding the management lag.

In subsequent years `pre` is taken from the `hcr` tracking metric for the current year, and the TAC carried in `ctrl` is first corrected for any amount borrowed or banked in the previous cycle before new banking/borrowing is evaluated.

Banking and borrowing are triggered as follows:

- **Borrowing**: if the (corrected) TAC falls more than `diff` below `pre` *and* the stock is healthy, `rate * tac` is added to the current TAC and recorded in `borrowing.isys`.
- **Banking**: if the (corrected) TAC rises more than `diff` above `pre` *and* the stock is healthy, `rate * tac` is subtracted from the current TAC and recorded in `banking.isys`.

Only annual management cycles (`frq == 1`) are currently supported; a non-annual `frq` raises an error.

## Value

A list with two elements:

`ctrl` The input FLasher::fwdControl object with the adjusted TAC stored in `ctrl$value`.

`tracking` The updated tracking object, with `borrowing.isys` and `banking.isys` metrics written for the assessment year `ay`.

## Author(s)

Iago MOSQUEIRA iago.mosqueira@wur.nl

## See Also

`mse::mp()`, `mse::mpCtrl()`, `mse::mseCtrl()`

## Examples

```
## Not run:
data(sol274)

# mpCtrl combining a hockey-stick HCR with 10% banking/borrowing
ctrl <- mpCtrl(
  est   = mseCtrl(method = perfect.sa),
  hcr   = mseCtrl(method = hockeystick.hcr,
            args = list(metric = "ssb", trigger = 42000,
```

```
                        output = "catch", target = 11000)),
  isys  = mseCtrl(method = bank_borrow.is,
            args = list(rate = 0.10, healthy = 2, diff = 0.05)))

run <- mp(om, control = ctrl, args = list(iy = 2021, fy = 2035))

plot(om, list(BaB = run))

# Inspect the TAC-setting steps from the tracking object
items <- c("year", "hcr", "banking.isys", "borrowing.isys", "isys", "fwd")
dcast(
  tracking(run)[metric %in% items[-1],
    .(data = mean(data)), by = .(year, metric)],
  year ~ metric, value.var = "data")[, ..items]

## End(Not run)
```

---

`buffer.hcr` *Buffer-based harvest control rule*

---

## Description

Implements a buffer harvest control rule (HCR) that adjusts management output according to the recent value of a stock or index metric relative to a set of threshold reference points.

Tools for visualizing a buffer-based harvest control rule (HCR), including the HCR curve itself and optional shaded management-tier bands.

## Usage

```
buffer.hcr(
  stk,
  ind,
  metric = "wmean",
  target = 1,
  width = 0.5,
  lim = max(target * 0.1, target - 2 * width),
  bufflow = max(lim * 1.5, target - width),
  buffupp = target + width,
  sloperatio = 0.15,
  initial,
  nyears = 4,
  dupp = NULL,
  dlow = NULL,
  all = TRUE,
  scale = FALSE,
  ...,
  args,
```

```
  tracking
)

plot_buffer.hcr(
  args,
  obs = "missing",
  alpha = 0.3,
 labels = c(lim = "Limit", bufflow = "Lower~buffer", buffupp = "Upper~buffer", metric =
    metric, output = output),
  metric = args$metric,
  output = "multiplier",
  xlim = buffupp * 1.5,
  ylim = scale * 1.5
)

buffer_bands(x)
```

## Arguments

| | |
|---|---|
| stk | A stock object, typically an FLStock, used together with ind to compute the HCR metric. |
| ind | An index object used when computing the selected metric. |
| metric | Character string giving the metric used for the x-axis and, when obs is an FLStock, the variable extracted from FLCore::metrics(). |
| target | Numeric. Target value of the metric. |
| width | Numeric. Half-width of the buffer around target. |
| lim | Numeric. Lower limit reference point for the metric. |
| bufflow | Numeric. Lower bound of the buffer zone. |
| buffupp | Numeric. Upper bound of the buffer zone. |
| sloperatio | Numeric. Controls the slope of output increases above buffupp. |
| initial | Initial output value used when no previous output is available. |
| nyears | Number of recent years over which the metric is averaged. |
| dupp | Optional upper bound on proportional increase in output. |
| dlow | Optional lower bound on proportional decrease in output. |
| all | Logical. If TRUE, change limits are always applied. |
| scale | Logical. If TRUE, do not recover previous output from tracking. |
| ... | Additional arguments passed to mse::selectMetric(). |
| args | A list of arguments defining the buffer HCR. Typically includes values such as lim, bufflow, buffupp, sloperatio, and optionally other elements used by the plotting method. If an object of class mseCtrl is supplied to plot_buffer.hcr(), its arguments are extracted using args(). |
| tracking | A tracking object updated with metric, decision, tier, and output values. |
| obs | Optional observed values to overlay on the HCR plot. Can be: |

- an FLStock object, in which case metric and output are extracted and plotted;
- a numeric value, in which case a single point is added on the HCR curve;
- omitted, in which case no observations are added.

alpha — Numeric. Alpha transparency used for observed points and paths.

labels — Labels for plot annotations. A named vector or list with names among lim, bufflow, buffupp, metric, and output. Supplied values override the defaults.

output — Character string giving the output used for the y-axis and, when obs is an FLStock, the variable extracted from FLCore::metrics().

xlim — Numeric. Maximum x-axis value for the HCR plot. By default set to buffupp * 1.5.

ylim — Numeric. Maximum y-axis value for the HCR plot.

x — An object for which args() returns a list containing at least the buffer-HCR parameters lim, bufflow, and buffupp.

## Details

buffer.hcr() computes a piecewise decision multiplier from the selected metric and applies it to the previous output value.

buffer_bands() creates shaded ggplot2 rectangles marking the critical, recovery, buffer, and above-target zones implied by the same thresholds.

plot_buffer.hcr() draws the HCR curve implied by the buffer rule and can optionally overlay observed values.

buffer_bands() returns ggplot2 layers that shade the management tiers defined by the HCR thresholds.

The buffer harvest control rule is defined by threshold values that partition the x-axis metric into zones:

- below lim: critical zone;
- between lim and bufflow: recovery zone;
- between bufflow and buffupp: buffer zone;
- above buffupp: above-target zone.

plot_buffer.hcr() evaluates the implied HCR multiplier across a sequence of metric values and draws the resulting curve, including annotations for the key thresholds.

buffer_bands() constructs semi-transparent shaded rectangles spanning these threshold intervals to support interpretation of HCR plots.

## Value

For buffer.hcr(), a list with components ctrl and tracking.

For buffer_bands(), a list of ggplot2 layers for shading HCR zones.

For plot_buffer.hcr(), a ggplot2 object.

For buffer_bands(), a list of ggplot2 components:

- a ggplot2::geom_rect() layer that draws shaded tier bands;
- a ggplot2::scale_fill_manual() layer that assigns colours to tiers.

### plot_buffer.hcr

`plot_buffer.hcr()` computes the HCR response curve from the supplied buffer rule parameters and returns a ggplot object. The plot includes:

- the HCR line;
- vertical reference markers for `lim`, `bufflow`, and `buffupp`;
- labels for the threshold points;
- optional observed points or trajectories.

If `obs` is an `FLStock` object, the function extracts the chosen `metric` and `output` values using `FLCore::metrics()` and overlays them on the HCR curve. If there is a single iteration, both points and a path are added, and first/last years are labelled.

If `obs` is numeric, a single red point is placed at the corresponding location on the HCR curve.

### buffer_bands

`buffer_bands()` extracts `lim`, `bufflow`, and `buffupp` from `args(x)` and defines four contiguous x-axis intervals. A fourth threshold, `target = 1.5 * buffupp`, is used as the upper bound for the final tier.

The returned rectangles extend from `y = 0` to `y = Inf`, so they occupy the full vertical plotting range.

### Assumptions and caveats

- Thresholds are assumed to satisfy `0 <= lim <= bufflow <= buffupp`.
- `buffer_bands()` adds a fill scale and may therefore override an existing fill scale in a ggplot object.
- In `plot_buffer.hcr()`, default values such as `xlim` and `ylim` depend on objects created after argument expansion and may rely on the surrounding evaluation framework.
- `plot_buffer.hcr()` currently uses internal helper functions such as `FLCore::spread()`, `args()`, and `FLCore::metrics()`, and assumes these are available.

### Author(s)

Iago Mosqueira, WMR; Richard Hillary, CSIRO

### See Also

`plot_buffer.hcr()`

`args()`, `FLCore::metrics()`, `ggplot2::geom_line()`, `ggplot2::geom_rect()`, `ggplot2::scale_fill_manual()`

### Examples

```
## Not run:
args <- list(lim = 0.4, bufflow = 1, buffupp = 2, sloperatio = 0.2)

# Plot the HCR curve
plot_buffer.hcr(args, labels = list(metric = "CPUE", output = "C~mult"))
```

```
# Add buffer bands to a custom ggplot
layers <- buffer_bands(hcr)
p <- ggplot(dat, aes(x = biomass, y = harvest)) + geom_line()
for (ly in layers) p <- p + ly
p

## End(Not run)
```

---

`depletion.hcr` *Depletion-based harvest control rule*

---

## Description

Implements a depletion-based harvest control rule (HCR) in which management output is scaled according to stock status relative to carrying capacity.

## Usage

```
depletion.hcr(
  stk,
  ind,
  metric = "ssb",
  mult = 1,
  hrmsy,
  K,
  trigger = 0.4,
  lim = 0.1,
  min = 1e-05,
  initial,
  dupp = NULL,
  dlow = NULL,
  all = TRUE,
  ...,
  args,
  tracking
)
```

## Arguments

| | |
|---|---|
| `stk` | A stock object, typically an `FLStock`, used together with `ind` to compute the selected metric. |
| `ind` | An FLQuant object containing metrics to be used by the HCR. |
| `metric` | Character string specifying the metric to use in the HCR. Passed to `mse::selectMetric()`. Defaults to `"ssb"`. |

| | |
|---|---|
| mult | Numeric multiplier applied to the harvest rate after scaling by the decision rule. Defaults to 1. |
| hrmsy | Numeric target harvest rate associated with MSY conditions. |
| K | Numeric carrying capacity or proxy scaling constant used to convert the metric into a depletion ratio. |
| trigger | Numeric depletion threshold above which the HCR applies the full harvest rate. Defaults to 0.4. |
| lim | Numeric depletion limit reference point below which the HCR applies the minimum multiplier. Defaults to 0.1. |
| min | Numeric minimum multiplier applied when depletion falls below lim. Defaults to 0.00001. |
| initial | Optional numeric initial output value used when interannual change constraints are applied in the first management year. |
| dupp | Optional numeric upper bound on proportional increase in output. If not NULL, output increases are capped at pre * dupp. |
| dlow | Optional numeric lower bound on proportional decrease in output. If not NULL, output decreases are floored at pre * dlow. |
| all | Logical. If TRUE, change constraints are always applied. If FALSE, they are only applied when the metric is below trigger. Defaults to TRUE. |
| ... | Additional arguments intended for downstream metric calculation. |
| args | A list of auxiliary framework arguments unpacked by FLCore::spread(). The function expects at least ay, iy, dy, mys, data_lag, management_lag, and it. |
| tracking | A tracking object updated throughout the mse::mp() evaluation. |

## Details

The rule uses the ratio of a selected stock metric to K as a depletion indicator and adjusts the harvest rate relative to hrmsy depending on whether the stock is above a trigger level, between a limit and trigger, or below the limit.

The function first computes the selected metric from stk and ind using mse::selectMetric(), restricted to the terminal data year dy. This metric is tracked under "metric.hcr".

Stock status is then expressed as the depletion ratio:

$$met/K$$

The decision multiplier is calculated as a piecewise function:

- if met / K >= trigger, the multiplier is 1;
- if lim <= met / K < trigger, the multiplier increases linearly from 0 at lim to 1 at trigger;
- if met / K < lim, the multiplier is set to min.

This multiplier is recorded in tracking under "decision.hcr".

The rule also classifies stock status into three tiers based on depletion:

- Tier 1: below lim;

- Tier 2: between `lim` and `trigger`;
- Tier 3: at or above `trigger`.

Tier membership is stored under `"tier.hcr"`.

A harvest-rate target is then computed as:

$$hrtarget = hrmsy \times dec \times mult$$

and management output is set as:

$$out = met \times hrtarget$$

This provisional output is stored under `"output.hcr"` before optional upper and lower change constraints are applied.

If `dupp` and/or `dlow` are supplied, the previous output value is recovered either from `initial` in the first management year or from the tracking object in later years. The provisional output is then capped and/or floored according to the specified proportional limits.

Finally, missing values are replaced with zero and the result is converted to an `FLasher::fwdControl()` object with one catch target per management year.

## Value

A named list with components:

- `ctrl`: an `fwdControl` object specifying the catch target for the management years in `mys`;
- `tracking`: the updated tracking object containing recorded metric, decision, tier, and output values.

## Change constraints

If `dupp` is supplied, increases in output can be capped. If `dlow` is supplied, decreases can be limited. When `all = TRUE`, these constraints are always applied. When `all = FALSE`, they are only applied when the metric is below the `trigger`, allowing unconstrained changes when stock status is at or above the trigger level.

## Assumptions and caveats

- The function assumes `spread(args[...])` creates `ay`, `iy`, `dy`, `mys`, and `it` in scope.
- Although `...` is declared, it is not currently passed on in the call to `mse::selectMetric()` in the implementation shown.
- The previous output retrieval uses `tracking[metric == 'hcr' & year == ay, data]`, which assumes a specific internal structure for `tracking`.
- `track(tracking, "decision.hcr", mys)` stores the decision for all management years, whereas `track(tracking, "tier.hcr", ay)` stores tier at assessment year resolution.
- Missing outputs are silently converted to zero.

## Author(s)

Iago Mosqueira, WMR

## See Also

buffer.hcr(), mse::selectMetric(), FLasher::fwdControl(),

---

effort.is *Effort-multiplier implementation system method*

---

## Description

Converts a fishing-mortality target set by an HCR into a relative effort multiplier by scaling against a reference fishing mortality computed from recent years. Passing a relative effort target to FLasher::fwd() rather than an absolute F smooths the response to inter-annual variability in fleet dynamics.

## Usage

```
effort.is(
  stk,
  ctrl,
  nyears = args$nsqy,
  Fref = yearMeans(fbar(stk)[, ac(seq(dy - nyears + 1, dy))]),
  args,
  tracking
)
```

## Arguments

| | |
|---|---|
| stk | An FLCore::FLStock object containing stock and fishery information. |
| ctrl | An FLasher::fwdControl object whose quant slot must be one of "f", "fbar", or "effort" and whose value slot carries the absolute F target from the HCR. |
| nyears | Integer. Number of years over which the mean reference fishing mortality is calculated. Defaults to args$nsqy as supplied by mse::mp(). |
| Fref | Numeric or FLCore::FLQuant. Reference fishing mortality used as the denominator when computing the multiplier. Defaults to the mean of FLCore::fbar() over the nyears years ending at dy (the final data year). |
| args | A list of management-cycle dimensionality arguments supplied automatically by mse::mp(), including at least dy and nsqy. |
| tracking | An FLCore::FLQuant used to record intermediate values and decisions during MP evaluation, passed through unchanged by this method. |

## Details

The effort multiplier is defined as

$$m = \frac{Ftarget}{F_ref}$$

where $F_target$ is the value in ctrl$value as set by the HCR and $F_ref$ is the mean FLCore::fbar() over the most recent nyears years of data.

Setting `ctrl$relYear` tells `FLasher::fwd()` to express the effort target relative to the effort observed in that reference year, effectively turning the absolute F recommendation into a proportional change instruction.

An error is raised immediately if `ctrl$quant` is not one of `"f"`, `"fbar"`, or `"effort"`.

## Value

A list with two elements:

`ctrl` The input FLasher::fwdControl object with `value` replaced by the dimensionless effort multiplier and `relYear` set to `ctrl$year - data_lag` so that `FLasher::fwd()` interprets the target relative to the correct historical year.

`tracking` The tracking object, returned unchanged.

## Author(s)

Iago MOSQUEIRA iago.mosqueira@wur.nl

## See Also

`mse::mp()`, `mse::split.is()`, `mse::mpCtrl()`, `mse::mseCtrl()`, `FLCore::fbar()`

## Examples

```
## Not run:
data(plesim)

# mpCtrl combining a hockey-stick HCR (F target) with effort.is
ctrl <- mpCtrl(
  est  = mseCtrl(method = perfect.sa),
  hcr  = mseCtrl(method = hockeystick.hcr,
           args = list(metric = "ssb", trigger = 45000,
                       output = "fbar", target = 0.27)),
  isys = mseCtrl(method = effort.is, args = list(nyears = 3)))

run      <- mp(om, control = ctrl,      args = list(iy = 2021, fy = 2035))
run_nois <- mp(om, control = ctrl[-3],  args = list(iy = 2021, fy = 2035))

# Compare with and without the effort.is buffer effect
plot(om, effort.is = run, no_is = run_nois)

## End(Not run)
```

`inspect` *Inspect MSE Tracking Data*

## Description

The `inspect` function turns the long `tracking` table into a year a metric summary on which to track the inoputs and outputs of all steps inside a call to the `mp` function.

## Usage

```
inspect(tab, metrics = NULL, summary = medmad)
```

## Arguments

| | |
|---|---|
| `tab` | An `FLms+e` object or a `data.table` containing tracking data. If an `FLmse` object is supplied, the `tracking` slot is extracted. |
| `metrics` | A character vector specifying the metrics to include in the output. If a single character value is given and it does not match excatly any of the contained metrics, it is used to subset using the datatable::%ilike% function. The special value `"decisions"` can be used to select all metrics from `"hcr"` onward. If `NULL`, the default, all metrics are returned. |
| `summary` | A function (such as `mean` or `median`) to summarize the `data` column within the tracking `data.table` across rthe `iter` dimension. Defaults to `medmad` which returns a string with `"Median (Median Absolute Deviation)"`. |

## Details

The function processes the table on the `tracking` slot by:

- Arranging metrics based on the order in which they are produced inside `mp()`.
- Subsets metrics based on user input or predefined criteria like `"decisions"`.
- Aggregates the `data` column using the `summary` function provided.
- Reshapes the processed `data.table` into a year-by-metrics format for easy reading.

The predefined metric order is that of the various steps inside `mp()`: `"om"`, `"obs"`, `"est"`, `"ind"`, `"phcr"`, `"hcr"`, `"isys"`, `"tm"`, `"iem"`, `"fb"`, `"fwd"`. Extra tracks produced inside any module are placed before the one named after the module that produced it.

## Value

A `data.table` where rows correspond to years and columns to selected metrics, with the tracking `data` aggregated using the specified summary function.

## Examples

```
# Load example MP run
data(mserun)
# Inspect all metrics
inspect(run)
# Inspect with different summary function
inspect(tracking(run), summary=mean)
# Inspect metrics with "SB" or "sb" in their names
inspect(tracking(run), metrics="SB")
# Inspect specific metrics
inspect(tracking(run), metrics=c("C.om", "C.obs", "C.est"))
# Inspect only decision-related metrics from 'hcr'
inspect(tracking(run), metrics="decisions")
```

---

# performanceFLQuants *Convert a performance statistics table to FLQuant objects*

---

## Description

Transforms a performance statistics `data.table`, or an `FLmse`/`FLmses` object, into one or more FLCore::FLQuant objects organized by management procedure, with statistics in the first (`quant`) dimension and iterations preserved in the `iter` dimension.

## Usage

```
performanceFLQuant(x)

performanceFLQuants(x)
```

## Arguments

| | |
|---|---|
| `x` | An object of class mse::FLmse, mse::FLmses, or a `data.table`. When `x` is an `FLmse` or `FLmses`, the performance slot is extracted automatically via `mse::performance()`. When `x` is a `data.table`, it must contain at least the columns `statistic`, `year`, `iter`, `data`, and `mp`. An optional `biol` column — present in outputs from `mse::mp()` on `FLombf` operating models — is mapped to the `unit` dimension of the resulting FLCore::FLQuant. |

## Details

`performanceFLQuants()` always returns a named list (one `FLQuant` per MP). `performanceFLQuant()` is a convenience wrapper that unwraps the list when only one MP is present.

Conversion from `data.table` to `FLQuant` is delegated to `FLCore::as.FLQuant()`, which expects the columns to map unambiguously onto the standard FLQuant dimensions. The `statistic` column populates the `quant` dimension and must therefore contain values meaningful as dimension names.

When a `biol` column is present, `data.table::setnames()` renames it to `unit` before passing to `FLCore::as.FLQuant()`, leaving the original table unmodified.

An informative error is raised if `x` is a `data.table` that does not contain all required columns (`statistic`, `year`, `iter`, `data`, `mp`).

## Value

`performanceFLQuants()` A named list of FLCore::FLQuant objects, one per unique `mp` value. The first (`quant`) dimension is named by `statistic`; `year` and `iter` dimensions correspond to simulation years and iterations. When a `biol` column is present its values populate the `unit` dimension.

`performanceFLQuant()` If the list contains exactly one element, that FLCore::FLQuant is returned directly; otherwise the full named list is returned.

## Functions

- `performanceFLQuants()`: Convenience wrapper returning a single FLCore::FLQuant directly when only one management procedure is present, or the full named list otherwise.

## Author(s)

Iago MOSQUEIRA iago.mosqueira@wur.nl

## See Also

`mse::performance()`, mse::FLmse, mse::FLmses, FLCore::FLQuant, `FLCore::as.FLQuant()`, `readPerformance()`, `writePerformance()`

## Examples

```
## Not run:
# From an FLmse object
flqs <- performanceFLQuants(mse_run)

# From a pre-read data.table (multiple MPs)
dat <- readPerformance()
flqs <- performanceFLQuants(dat)
flqs[["MP1"]]["SSB", , , , ,]

# Convenience wrapper — returns FLQuant directly for a single MP
flq <- performanceFLQuant(dat)

## End(Not run)
```

## pid.hcr *Proportional-Integral-Derivative (PID) Harvest Control Rule (HCR)*

### Description

Implements a Proportional-Integral-Derivative (PID) based harvest control rule for adjusting Total Allowable Catch (TAC) based on divergence from a reference point using PID control signals.

### Usage

```
pid.hcr(
  stk,
  ind,
  ref,
  metric = ssb,
  initial,
  kp = 0,
  ki = 0,
  kd = 0,
  nyears = 5,
  dlow = NA,
  dupp = NA,
  args,
  tracking,
  ...
)
```

### Arguments

| | |
|---|---|
| stk | FLStock. The stock object to which the HCR applies. |
| ind | FLQuant. The abundance index used to compute the control signal. |
| ref | Numeric. The reference value for the metric to determine divergence. |
| metric | Character or function. The metric applied to measure stock status. Default is 'ssb' (spawning stock biomass). |

## shortcut.sa *Shortcut stock assessment including error on a status metric*

### Description

Performs a simplified or "shortcut" stock assessment by truncating an operating model stock object to the current data year, computing a selected stock metric, applying stochastic deviations, and returning the resulting indicator together with the truncated stock and updated tracking object.

## Usage

```
shortcut.sa(
  stk,
  idx,
  metric = "ssb",
  SSBdevs = met %=% 1,
  devs = SSBdevs,
  args,
  tracking,
  ...
)
```

## Arguments

| | |
|---|---|
| stk | An FLStock-like object representing the stock available to the shortcut assessment. The object is truncated to the terminal data year dy. |
| idx | An index object. Currently included for interface compatibility with other assessment functions, but not used directly inside this function. |
| metric | A character string naming the function used to compute the assessment metric from stk. Defaults to "ssb". The named function is called via base::do.call(). |
| SSBdevs | An object containing deviations to apply to the computed metric. Defaults to met %=% 1, i.e. a unit deviation structure matching the dimensions of the computed metric. Typically this is an FLQuant or compatible object. |
| devs | Deprecated alias or shorthand for SSBdevs. Defaults to SSBdevs. |
| args | A list-like object containing dimensions and assessment settings needed by FLCore::spread(). In particular, this function assumes that variables such as dy, y0, ay, and stock become available after calling spread(args). |
| tracking | A tracking object updated in-place through to record convergence of the shortcut assessment. |
| ... | Additional arguments passed to the metric function named in metric. |

## Details

This function is intended for use inside management procedure workflows where a full assessment model is not run, but an index or assessment quantity is approximated from the operating model using a direct transformation of stock state and observation-error-like deviations.

The function proceeds as follows:

1. values in args are unpacked using FLCore::spread();
2. the stock object is truncated to end=dy using window();
3. the requested metric is computed by calling the function named in metric on stk;
4. the result is collapsed across units using FLCore::unitSums();
5. stochastic deviations are applied over the period from y0 to dy;
6. the resulting quantity is wrapped in an FLCore::FLQuants() object and named with the selected metric;

7. convergence is recorded in `tracking` by setting `"conv.est"` to `1` for year `ay` and biological unit `stock`.

## Value

A named list with components:

- `stk`: the stock object truncated to the terminal data year `dy`;
- `ind`: an `FLQuants` object containing the derived indicator series, named according to `metric`;
- `tracking`: the updated tracking object with convergence information recorded.

## Assumptions

- the function named in `metric` exists and accepts `stk` as its first argument;
- `devs` or `SSBdevs` are dimensionally compatible with the metric computed from `stk`;

## Notes

- `idx` is currently unused but may be retained for consistency with other assessment-method signatures.
- `devs` is currently used to build the indicator, even though `SSBdevs` is the more explicit argument name.

## Author(s)

Iago Mosqueira, WMR; Ernesto Jardim, IPMA

## See Also

`shortcut_devs()`, `FLCore::ssb()`

## Examples

```
## Not run:
# dataset contains both OM (FLom) and OEM (FLoem)
data(plesim)
# Create shortcut deviances for F and SSB
devs <- shortcut_devs(om, Fcv = 0.3, Fphi = 0.6)
# Set control: sa and hcr
control <- mpCtrl(list(
  est = mseCtrl(method=shortcut.sa,
   args=list(devs=devs$SSB)),
  hcr = mseCtrl(method=hockeystick.hcr, args=list(lim=0,
  trigger=14000, target=0.18))))
# Runs mp
tes <- mp(om, ctrl=control, args=list(iy=2021, fy=2026))

## End(Not run)
```

shortcut_devs *Generate deviation series for shortcut assessments*

## Description

Creates stochastic multiplicative deviation series for use in shortcut stock assessments, typically representing uncertainty or observation error applied to fishing mortality and spawning stock biomass metrics.

## Usage

```
shortcut_devs(
  om,
  Fcv = 0.212,
  Fphi = 0.423,
  SSBcv = 0,
  SSBphi = 0,
  bias.correct = FALSE
)
```

## Arguments

| | |
|---|---|
| om | An operating model object used only to define dimensions for the generated deviation series. The function uses `dims(om)$iter` for the number of iterations and `dimnames(om)$year` for the year dimension. |
| Fcv | Numeric. Standard deviation on the log scale for the fishing mortality deviation process. Defaults to `0.212`. |
| Fphi | Numeric. Autocorrelation parameter for the fishing mortality deviation process. Defaults to `0.423`. |
| SSBcv | Numeric. Standard deviation on the log scale for the spawning stock biomass deviation process. Defaults to `0`. |
| SSBphi | Numeric. Autocorrelation parameter for the spawning stock biomass deviation process. Currently not used in the implementation, which fixes the SSB autocorrelation at `0`. Defaults to `0`. |
| bias.correct | Logical. If `TRUE`, applies bias correction when generating the lognormal AR(1) deviates. Passed to `FLCore::rlnormar1()`. Defaults to `FALSE`. |

## Details

The deviations are generated as lognormal autoregressive series and returned as an `FLQuants` object with components for `F` and `SSB`.

The function generates deviations using `FLCore::rlnormar1()`, with:

- `n` = `dims(om)$iter` iterations;
- `years` = `dimnames(om)$year`;

- `meanlog = 0`, so the process is centered on multiplicative value 1 on the log scale before optional bias correction;
- separate variance and autocorrelation settings for `F` and `SSB`.

The returned object is suitable for direct use in `shortcut.sa()`, where the deviations can be applied to derived stock metrics.

## Value

An `FLQuants` object with two elements:

- `F`: a lognormal AR(1) deviation series for fishing mortality;
- `SSB`: a lognormal deviation series for spawning stock biomass.

## Current implementation note

Although `SSBphi` is provided as an argument, the function currently hardcodes `rho=0` for the `SSB` process. This means the SSB deviations are independent through time regardless of the value supplied to `SSBphi`.

## Author(s)

Iago Mosqueira, WMR; Ernesto Jardim, IPMA

## See Also

`shortcut.sa()`, `FLCore::rlnormar1()`, `FLCore::FLQuants()`

## Examples

```
## Not run:
# Generate default shortcut deviations
devs <- shortcut_devs(om)

# More variable and autocorrelated F deviations
devs <- shortcut_devs(om, Fcv = 0.3, Fphi = 0.6)

# With bias correction
devs <- shortcut_devs(om, bias.correct = TRUE)

## End(Not run)
```

trackingFLQuant *Convert Performance Data to FLQuant Format*

## Description

Transforms performance metrics from FLmse or FLmsea objects, stored as a data.table into an FLQuant pr FLQuants object.

## Usage

```
trackingFLQuant(x)

trackingFLQuants(x)
```

## Arguments

| | |
|---|---|
| x | An object of class FLmse, FLmses, or a data.table. |

## Details

If x is of class FLmse or FLmses, the performance slot is extracted automatically. The data.table must contain columns: statistic, year, iter, data, and mp. Optionally, a biol column can be present, as in the performance table obtained from callin mp() on an FLombf OM. Values for each biol will be separated using the 'unit' dimension.

The first (quant) dimension of the FLQuant or FLQuants contain the statistics, named as in the statistic column of the performance table.

When results from multiple management procedures are present, the output is an FLQuants list of FLQuant objects, one per mp.

## Value

If a single management procedure (mp) is present, returns a single FLQuant object. If multiple management procedures are present, returns a list of FLQuant objects, one per mp. When biol column is present, it is renamed to unit in the resulting FLQuant or FLQuants.

## See Also

mse::performance(), mse::FLmse, FLmses

## Examples

```
## Not run:
  # Using a data.table directly
  perf_dt <- data.table(
    statistic = rep("SSB", 4),
    year = rep(2025:2026, 2),
    iter = rep(1:2, each = 2),
    data = runif(4, 1000, 3000),
```

```
    mp = "MP1"
  )
  quant <- trackingFLQuant(perf_dt)
  quants <- trackingFLQuants(perf_dt)

## End(Not run)
```

---

`writePerformance` *Read and write performance statistics tables*

---

## Description

A family of utilities for persisting, retrieving, labelling, summarizing, and reshaping performance statistics tables produced by `mse::performance()`.

## Usage

```
validatePerformance(dat)

hasPerformance(file = "model/performance.dat.gz", om = NULL, type = NULL, run = NULL)

listPerformance(file = "model/performance.dat.gz")

deletePerformance(
  file = "model/performance.dat.gz",
  om = NULL,
  type = NULL,
  run = NULL,
  mp = NULL,
  dry_run = FALSE
)

diffPerformance(dat, file = "model/performance.dat.gz")

writePerformance(dat, file = "model/performance.dat.gz", overwrite = FALSE)

readPerformance(file = "model/performance.dat.gz")

summaryPerformance(file = "model/performance.dat.gz")

labelPerformance(dat, labels = NULL)

setLabelPerformance(file = "model/performance.dat.gz", labels)

periodsPerformance(x, periods)
```

```
extractPerformance(dat, mp)

getOMPerformance(path, pattern = "*.rds", fy, ...)

getMSEPerformance(path, pattern = "*.rds")
```

## Arguments

| | |
|---|---|
| dat | A data.table containing performance statistics, as returned by mse::performance(), or an object of class FLmse or FLmses from which performance statistics can be extracted automatically. |
| file | A character string giving the path to the performance table file. Defaults to "model/performance.dat.gz". The .gz extension causes data.table::fwrite() / data.table::fread() to use gzip compression transparently. |
| overwrite | Logical. If TRUE, any existing file at file is overwritten entirely. If FALSE (default), existing rows not conflicting with dat are preserved and new rows are appended; conflicts are resolved by an anti-join on biol, statistic, year, iter, om, type, and run. |
| om, type, run | Character scalar identifiers used to filter the stored table. Any combination may be supplied; unspecified arguments are ignored. |
| labels | The labelling specification used by labelPerformance() and setLabelPerformance(). Accepted forms:<br>NULL **(default)** Each row is labelled by its mp value if one is present, or by om otherwise.<br>"numeric" Management procedures are assigned sequential labels "MP1", "MP2", ... in the order they appear.<br>**A named** list Names are matched against mp (or om) and the values are used as labels.<br>**A** data.frame **or** data.table Must contain columns element and label; element is matched against mp (or om). |
| x | A data.table containing performance statistics for periodsPerformance(), as returned by mse::performance() or readPerformance(). Must contain at least year, data, type, mp, statistic, name, desc, and iter. |
| periods | A vector or list whose elements define the years belonging to each period (used by periodsPerformance()). Named elements use the name as the period label; unnamed elements are labelled automatically from the year range (e.g. "2026-35"). |
| mp | A character value (or vector) used by extractPerformance(). Length-1 values are matched with data.table::like() (%like%) so partial strings and regular-expression patterns are accepted; vectors are matched exactly with %in%. |
| path | A character string giving the directory containing serialized files, used by getOMPerformance() and getMSEPerformance(). |
| pattern | A character string passed to base::list.files() to filter which files are read. Defaults to "*.rds". |

| | |
|---|---|
| fy | The final year to which each operating model is trimmed with FLCore::window() before performance is computed (getOMPerformance() only). |
| ... | Additional arguments forwarded to mse::performance() (getOMPerformance() only). |
| dry_run | Logical. If TRUE (default FALSE), return the rows that would be deleted without writing any changes. |

## Details

validatePerformance() Checks required columns, numeric types, NA values in key columns, and key uniqueness; stops or warns on failure.

hasPerformance() Returns TRUE if the file contains rows matching the supplied identifiers, without loading the full table.

listPerformance() Returns a catalogue data.table of (om, type, run, mp) combinations with row counts and year ranges.

deletePerformance() Deletes rows matching supplied identifiers; supports a dry_run preview mode.

diffPerformance() Compares an in-memory table against the stored file and categorises rows as new, replace, or unchanged.

writePerformance() Serializes a table to disk, merging with any existing file.

readPerformance() Reads a stored table and restores column classes, key, and factor levels.

summaryPerformance() Prints a compact console summary.

labelPerformance() Adds or replaces the label column.

setLabelPerformance() Reads, re-labels, and overwrites a stored table.

periodsPerformance() Aggregates annual statistics into user-defined periods.

extractPerformance() Subsets to selected MPs together with their OM baseline rows.

getOMPerformance() Batch-reads OM files and returns a combined performance table.

getMSEPerformance() Batch-reads MSE result files and returns a combined performance table.

writePerformance

Before writing, all columns are coerced to character and then year and data are restored to numeric to ensure a consistent serialization format. Optional columns are normalized: if neither type nor run is present, empty character columns for type, run, and mp are added; if mp is absent but type and run are present, mp is constructed by pasting om, type, and run with "_"; if label is absent it is set to mp when an MP is present, otherwise to om. Column order is standardized to om, type, run, mp, biol, statistic, name, desc, year, iter, data before writing.

readPerformance

Reading is performed with data.table::fread() with explicit colClasses to avoid ambiguous type inference (e.g. when iter contains non-numeric labels or mp is an empty string). The key set by data.table::setkey() enables efficient subsetting and is assumed by several downstream functions. The label column is optional; when absent it is not created — use labelPerformance() to add it after reading.

periodsPerformance

periods is coerced to a list internally. The compact year label uses only the last two digits of the final year (2026:2035 → "2026-35"). Missing values in data are silently ignored (na.rm = TRUE). When x contains a label column the grouping includes it; otherwise grouping is by type, mp, statistic, name, desc, and iter.

extractPerformance

Operating model baseline rows are identified by an empty string in mp. For scalar mp, matching uses %like%; use anchored patterns (e.g. "^HCR1$") when MP names share substrings. For vectors, exact matching (%in%) is used.

getOMPerformance

Each .rds file is expected to contain a list-like object with an $om component compatible with FLCore::window() and mse::performance(). Warnings from mse::performance() are suppressed with base::suppressWarnings(); re-run individual files interactively if unexpected results appear.

getMSEPerformance

If a file's deserialized object already inherits from data.table it is returned directly, allowing precomputed performance summaries to be mixed with raw MSE objects. fill = TRUE in data.table::rbindlist() tolerates minor column-structure differences across files.

## Value

writePerformance(), setLabelPerformance() Invisibly TRUE.

deletePerformance() Invisibly, a data.table of the deleted rows. When dry_run = TRUE, the rows that would be deleted are returned visibly without modifying the file.

diffPerformance() A named list with elements new, replace, and unchanged, each a data.table.

readPerformance() A data.table keyed by om, type, run, biol, mp, statistic, and year, with grouping columns as factors and column order fixed as om, type, run, mp, biol, statistic, name, desc, year, iter, data.

summaryPerformance() Invisibly TRUE (called for its printed side-effect).

labelPerformance() A copy of dat with a label factor column added or replaced; OM-only rows appear first in the factor levels.

periodsPerformance() A data.table with one row per grouping combination and period, containing period, year (compact range string), data (period mean), and the inherited grouping columns.

extractPerformance() A subset of dat containing the matched MP rows and their corresponding OM baseline rows (mp == "").

getOMPerformance(), getMSEPerformance() A data.table produced by row-binding the performance results from all matching files.

## Functions

- validatePerformance(): Validate a performance statistics table, checking for required columns, correct numeric types, NA values in key columns, and duplicate rows on the composite primary key.
- hasPerformance(): Test whether a stored performance table contains rows matching the supplied identifiers, without reading the full table into memory.

- `listPerformance()`: Return a compact catalogue of the `(om, type, run, mp)` combinations stored in a performance file, together with row count, year range, and number of distinct iterations — without loading the `data` column.
- `deletePerformance()`: Delete rows from a stored performance table that match the supplied identifiers. When `dry_run = TRUE` the rows that *would* be deleted are returned without modifying the file.
- `diffPerformance()`: Compare an in-memory performance table against a stored file and categorise every row as *new*, *replace* (key exists in file), or *unchanged* (key exists and values are identical). Returns a named list of three `data.tables`; the file is not modified.
- `readPerformance()`: Read a performance statistics table from disk, restoring column classes, data.table key, column order, and factor levels.
- `summaryPerformance()`: Print a compact console summary of the number of operating models, MP types, and management procedures, together with per-group year ranges and iteration counts.
- `labelPerformance()`: Add or replace the `label` factor column in a performance statistics table; OM-only rows appear first in the factor levels.
- `setLabelPerformance()`: Convenience wrapper that reads a stored performance table, updates its `label` column via `labelPerformance()`, and writes the result back, replacing the previous contents.
- `periodsPerformance()`: Collapse annual performance statistics into broader user-defined periods by computing the mean of `data` over the years belonging to each period.
- `extractPerformance()`: Subset a performance table to the rows belonging to management procedures matching a pattern, together with their corresponding OM baseline rows (`mp == ""`).
- `getOMPerformance()`: Batch-read serialized OM files, trim each to `fy` with `FLCore::window()`, compute `mse::performance()`, and row-bind the results into a single table.
- `getMSEPerformance()`: Batch-read serialized MSE result files or pre-computed performance `data.table` files and row-bind them into a single table.

## Author(s)

Iago MOSQUEIRA iago.mosqueira@wur.nl

## See Also

`mse::performance()`, mse::FLmse, mse::FLmses, `FLCore::window()`, `performanceFLQuants()`

## Examples

```
## Not run:
## writePerformance / readPerformance
writePerformance(perf_dat)
dat <- readPerformance()
dat <- readPerformance("results/performance.dat.gz")

## validatePerformance
validatePerformance(perf_dat)
```

```
## hasPerformance
hasPerformance()
hasPerformance(om = "ple.27.420")
if(!hasPerformance(om = "ple.27.420", run = "r01"))
  writePerformance(perf_dat)

## listPerformance
listPerformance()
listPerformance("results/performance.dat.gz")

## diffPerformance
d <- diffPerformance(perf_dat)
d$replace
if(nrow(d$replace) == 0) writePerformance(perf_dat)

## deletePerformance
deletePerformance(om = "ple.27.420", run = "r01", dry_run = TRUE)
deletePerformance(om = "ple.27.420", run = "r01")
deletePerformance(mp = "hcr_Fmsy")

## summaryPerformance
summaryPerformance()
summaryPerformance(dat)

## labelPerformance
dat <- labelPerformance(dat)
dat <- labelPerformance(dat, labels = "numeric")
dat <- labelPerformance(dat, labels = list(
  hcr01 = "Conservative HCR", hcr02 = "Moderate HCR"))

## setLabelPerformance
setLabelPerformance(labels = "numeric")
setLabelPerformance("results/performance.dat.gz",
  labels = list(hcr01 = "Low F", hcr02 = "High F"))

## periodsPerformance
res <- periodsPerformance(dat, periods = list(
  short  = 2021:2025,
  medium = 2026:2035,
  long   = 2036:2050))

## extractPerformance
sub <- extractPerformance(dat, "HCR1")
sub <- extractPerformance(dat, "^HCR1$")

## getOMPerformance
dat <- getOMPerformance("om", fy = 2025)
dat <- getOMPerformance("om", fy = 2025, probs = c(0.1, 0.5, 0.9))

## getMSEPerformance
dat <- getMSEPerformance("mse")
dat <- getMSEPerformance("mse", pattern = "scenario_.*\\.rds$")
```

```
## End(Not run)
```

# Index