Package 'functionals'

November 3, 2026

Type Package

Title Functional Programming with Parallelism and Progress Tracking

Version 0.6.0

Maintainer Imad El Badisy <elbadisyimad@gmail.com>

Description Provides functional tools such as fmap(), fwalk(), and fapply() to iterate over vectors, data frames, or grouped data with optional parallelism and real-time progress tracking. Progress updates reflect completed tasks across sequential, multicore, and cluster-backed execution. A uniform error policy keeps a run going when individual tasks fail, and optional per-task L'Ecuyer-CMRG random-number streams make parallel Monte Carlo work reproducible regardless of the number of cores. Designed for readable and reproducible workflows, including simulation and benchmarking.

License MIT + file LICENSE

Encoding UTF-8

Imports parallel

Suggests testthat (>= 3.0.0), bench, knitr, purrr, rmarkdown, rsample

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.3

URL https://CRAN.R-project.org/package=functionals

BugReports https://github.com/ielbadisy/functionals/issues

NeedsCompilation no

Author Imad El Badisy [aut, cre]

Repository CRAN

Date/Publication 2026-11-03 09:30:02 UTC

Contents

fapply 2
fcompose 4
fcv 5
floop 6
fmap 7
fmapc 8
fmapg 9
fmapn 10
fmapr 12
freduce 13
frepeat 14
fwalk 16

Index **18**

`fapply` *Apply a function over a list or vector with optional parallelism and progress*

Description

A lightweight and fast version of ‘lapply()‘ with support for multicore (Unix) and snow-style clusters via ‘parallel‘, with internal progress bar tracking and message suppression. When ‘pb = TRUE‘, progress advances when individual tasks complete, including in the parallel paths, so counts reflect completed jobs rather than internal chunk boundaries. Progress rendering is throttled for large workloads to keep console redraw overhead bounded, while status output reports elapsed time and estimated time remaining.

Usage

```
fapply(
  .x,
  .f,
  ncores = 1,
  pb = FALSE,
  cl = NULL,
  load_balancing = TRUE,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
  .seed = NULL,
  ...
)
```

Arguments

.x	A list or atomic vector.
.f	Function to apply.
ncores	Number of cores to use (default: 1 = sequential).
pb	Show progress bar? (default: FALSE). When enabled, progress is driven by completed tasks. In parallel mode, updates occur as workers return results.
cl	A cluster object (from parallel::makeCluster), or integer for core count.
load_balancing	Logical. Use ‘parLapplyLB‘ if ‘TRUE‘ (default: ‘FALSE‘).
.on_error	How to handle an error thrown by ‘.f‘. ‘"stop"‘ (default) aborts the whole run, matching base ‘lapply()‘. ‘"pass"‘ keeps going and stores an object of class ‘"functionals_error"‘ in the failed position (inspect with ‘conditionMessage()‘), then emits a one-line summary warning. ‘"fill"‘ replaces each failed result with ‘.fill‘. The behaviour is identical whether execution is sequential or parallel.
.fill	Replacement value used for failed elements when ‘.on_error = "fill"‘. Default ‘NULL‘.
.seed	Optional single number. When supplied, task ‘i‘ is run with its own L’Ecuyer-CMRG random-number stream, so results are reproducible and independent of ‘ncores‘ or task-completion order. The caller’s global RNG state is restored on exit.
...	Additional arguments passed to ‘.f‘.

Value

A list of results.

Examples

```
# Basic usage (sequential)
fapply(1:5, sqrt)

# With progress bar (sequential)
fapply(1:5, function(x) { Sys.sleep(0.1); x^2 }, pb = TRUE)

# Multicore on Unix (if available)

if (.Platform$OS.type != "windows") {
  fapply(1:10, sqrt, ncores = 2)
}


# With user-created cluster (portable across platforms)

cl <- parallel::makeCluster(2)
fapply(1:10, sqrt, cl = cl)
parallel::stopCluster(cl)
```

```
# Heavy computation example with exact completion-driven progress

heavy_fn <- function(x) { Sys.sleep(0.05); x^2 }
fapply(1:20, heavy_fn, ncores = 2, pb = TRUE)
```

`fcompose` *Compose multiple functions*

Description

Create a new function by composing several functions, applied from right to left. Equivalent to 'f1(f2(f3(...)))'.

Usage

```
fcompose(...)
```

Arguments

`...`	Functions to compose. Each must take a single argument and return an object compatible with the next function in the chain.

Value

A new function equivalent to nested application of the input functions.

Examples

```
square <- function(x) x^2
add1 <- function(x) x + 1

f <- fcompose(sqrt, square, add1)  # => sqrt(square(x + 1))
f(4)  # => sqrt((4 + 1)^2) = sqrt(25) = 5

# More compact
fcompose(log, exp)(2)  # log(exp(2)) = 2
```

`fcv` *Functional Cross-Validation mapping*

Description

Applies a user-defined function ‘.f‘ to each element of ‘.splits‘, typically from cross-validation objects such as ‘rsample::vfold_cv()‘.

Usage

```
fcv(
  .splits,
  .f,
  ncores = NULL,
  pb = FALSE,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
  .seed = NULL,
  ...
)
```

Arguments

`.splits`	A list of resample splits (e.g., from ‘rsample::vfold_cv()‘).
`.f`	A function to apply to each split. Typically expects a single ‘split‘ object.
`ncores`	Integer. Number of cores to use for parallel processing. Default is ‘NULL‘ (sequential).
`pb`	Logical. Whether to display a progress bar. Default is ‘FALSE‘.
`.on_error`	How errors thrown by ‘.f‘ are handled: ‘"stop"‘ (default), ‘"pass"‘, or ‘"fill"‘. See [fapply()].
`.fill`	Replacement for failed folds when ‘.on_error = "fill"‘.
`.seed`	Optional single number for reproducible per-fold RNG streams.
`...`	Additional arguments passed to ‘.f‘.

Details

‘fcv()‘ is a thin, intention-revealing alias for [fapply()] specialised to a list of resample splits: ‘fcv(splits, f)‘ is exactly ‘fapply(splits, f)‘. Use it when a call site reads better as "cross-validate" than as "apply".

Value

A list of results returned by applying ‘.f‘ to each element of ‘.splits‘.

Examples

```
if (requireNamespace("rsample", quietly = TRUE)) {
  set.seed(123)
  cv_splits <- rsample::vfold_cv(mtcars, v = 5)

  # Apply summary over training sets
  fcv(cv_splits$splits, function(split) {
    summary(rsample::analysis(split))
  })

  # With progress and parallel execution

    fcv(cv_splits$splits, function(split) {
      summary(rsample::analysis(split))
    }, ncores = 2, pb = TRUE)

}
```

`floop` *Functional loop with optional parallelism and progress bar*

Description

Deprecated in 0.6.0. ‘floop()‘ never did anything that [fmap()] and [fwalk()] do not: ‘floop(.x, .f)‘ is [fmap()], and ‘floop(.x, .f, .capture = FALSE)‘ is [fwalk()]. Use those directly. ‘floop()‘ will be removed in a future release.

Usage

```
floop(.x, .f, ncores = 1, pb = FALSE, .capture = TRUE, ...)
```

Arguments

`.x`	A vector or list of elements to iterate over.
`.f`	A function to apply to each element of ‘.x‘.
`ncores`	Integer. Number of cores to use. Default is 1 (sequential).
`pb`	Logical. Show a progress bar? Default is ‘FALSE‘.
`.capture`	Logical. If ‘TRUE‘ (default) results are returned (like [fmap()]); if ‘FALSE‘, ‘.x‘ is returned invisibly (like [fwalk()]).
`...`	Additional arguments passed to ‘.f‘.

Value

A list of results if ‘.capture = TRUE‘, otherwise ‘.x‘ invisibly.

See Also

[fmap()], [fwalk()]

Examples

```
# Prefer fmap() / fwalk():
fmap(1:3, function(i) i^2)
```

`fmap` *Functional mapping with optional parallelism and progress bars*

Description

Applies a function ‘.f‘ to each element of ‘.x‘, with optional parallel processing and progress bar support. Progress behavior is inherited from ‘fapply()‘, including completion-driven updates in the parallel execution paths, bounded redraw frequency for large workloads, and elapsed/ETA status reporting.

Usage

```
fmap(
  .x,
  .f,
  ncores = NULL,
  pb = FALSE,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
  .seed = NULL,
  ...
)
```

Arguments

`.x`	A list or atomic vector of elements to iterate over.
`.f`	A function to apply to each element of ‘.x‘. Can be a function or a string naming a function.
`ncores`	Integer. Number of CPU cores to use for parallel processing. Default is ‘NULL‘ (sequential).
`pb`	Logical. Whether to show a progress bar. Default is ‘FALSE‘. In parallel mode, progress updates as each completed task returns.
`.on_error`	How errors thrown by ‘.f‘ are handled: ‘"stop"‘ (default), ‘"pass"‘ (keep going, store a ‘"functionals_error"‘ object), or ‘"fill"‘ (replace with ‘.fill‘). See [fapply()].
`.fill`	Replacement for failed elements when ‘.on_error = "fill"‘.
`.seed`	Optional single number for reproducible per-task RNG streams.
`...`	Additional arguments passed to ‘.f‘.

Value

A list of results, one for each element of ‘.x‘.

Examples

```
slow_fn <- function(x) { Sys.sleep(0.01); x^2 }
x <- 1:100

# Basic usage
fmap(x, slow_fn)

# With progress bar
fmap(x, slow_fn, pb = TRUE)

# With parallel execution (non-Windows)

if (.Platform$OS.type != "windows") {
  fmap(x, slow_fn, ncores = 2, pb = TRUE)
}
```

---

fmapc *Apply a function column-wise with name access and parallelism*

Description

Applies a function ‘.f‘ to each column of a data frame ‘.df‘. Each call receives both the column vector and its name, enabling name-aware column processing. Supports parallel execution and progress display.

Usage

```
fmapc(
  .df,
  .f,
  ncores = NULL,
  pb = FALSE,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
  .seed = NULL,
  ...
)
```

Arguments

.df	A data frame whose columns will be iterated over.
.f	A function that takes two arguments: the column vector and its name.

ncores	Integer. Number of cores to use for parallel processing. Default is ‘NULL‘ (sequential).
pb	Logical. Whether to display a progress bar. Default is ‘FALSE‘.
.on_error	How errors thrown by ‘.f‘ are handled: ‘"stop"‘ (default), ‘"pass"‘, or ‘"fill"‘. See [fapply()].
.fill	Replacement for failed columns when ‘.on_error = "fill"‘.
.seed	Optional single number for reproducible per-column RNG streams.
...	Additional arguments passed to ‘.f‘.

Value

A list of results obtained by applying ‘.f‘ to each column of ‘.df‘.

Examples

```
df <- data.frame(a = 1:3, b = 4:6)

# Apply a function that returns column mean and name
fmapc(df, function(x, name) {
  list(mean = mean(x), var = var(x), name = name)
})

# With progress and parallel execution

fmapc(df, function(x, name) mean(x), ncores = 2, pb = TRUE)
```

---

fmapg *Apply a function to groups of a data frame in parallel*

Description

Applies a function ‘.f‘ to each group of rows in a data frame ‘.df‘, where grouping is defined by one or more variables in ‘by‘. Each group is passed as a data frame to ‘.f‘. Supports parallelism and optional progress display.

Usage

```
fmapg(
  .df,
  .f,
  by,
  ncores = NULL,
  pb = FALSE,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
```

```
  .seed = NULL,
  ...
)
```

Arguments

.df	A data frame to group and apply the function over.
.f	A function to apply to each group. The function should accept a data frame (a group).
by	A character vector of column names in '.df' used for grouping.
ncores	Integer. Number of cores to use for parallel processing. Default is 'NULL' (sequential).
pb	Logical. Whether to show a progress bar. Default is 'FALSE'.
.on_error	How errors thrown by '.f' are handled: '"stop"' (default), '"pass"', or '"fill"'. See [fapply()].
.fill	Replacement for failed groups when '.on_error = "fill"'.
.seed	Optional single number for reproducible per-group RNG streams.
...	Additional arguments passed to '.f'.

Value

A list of results, one for each group defined by 'by'.

Examples

```
# Group-wise mean of Sepal.Length in iris dataset
fmapg(iris, function(df) mean(df$Sepal.Length), by = "Species")

# Group-wise model fitting with progress and parallelism

fmapg(mtcars, function(df) lm(mpg ~ wt, data = df),
      by = "cyl", ncores = 2, pb = TRUE)
```

---

fmapn *Apply a function over multiple argument lists in parallel*

Description

Applies a function '.f' over multiple aligned lists in '.l'. Each element of '.l' should be a list or vector of the same length. Each call to '.f' receives one element from each list. Supports parallel execution and progress display.

Usage

```
fmapn(
  .l,
  .f,
  ncores = NULL,
  pb = FALSE,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
  .seed = NULL,
  ...
)
```

Arguments

.l	A list of vectors or lists. All elements must be of equal length.
.f	A function to apply. It must accept as many arguments as there are elements in '.l'.
ncores	Integer. Number of cores to use for parallel processing. Default is 'NULL' (sequential).
pb	Logical. Whether to display a progress bar. Default is 'FALSE'.
.on_error	How errors thrown by '.f' are handled: '"stop"' (default), '"pass"', or '"fill"'. See [fapply()].
.fill	Replacement for failed tuples when '.on_error = "fill"'.
.seed	Optional single number for reproducible per-tuple RNG streams.
...	Additional arguments passed to '.f'.

Value

A list of results obtained by applying '.f' to each tuple from '.l'.

Examples

```
# Fit a linear model for each response variable using the same predictor
df <- data.frame(
  y1 = rnorm(100),
  y2 = rnorm(100),
  x = rnorm(100)
)

# List of formulas and data
formulas <- list(y1 ~ x, y2 ~ x)
data_list <- list(df, df)

fmapn(list(formula = formulas, data = data_list),
      function(formula, data) lm(formula, data = data))

# Extract model summaries in parallel
models <- fmapn(
```

```
  list(formula = formulas, data = data_list),
  function(formula, data) summary(lm(formula, data = data))$r.squared
)
```

`fmapr` *Apply a function row-wise on a data frame with parallelism*

Description

Applies a function ‘.f‘ to each row of a data frame ‘.df‘, with optional parallelism and progress bar. Each row is converted to a named list before being passed to ‘.f‘, enabling flexible access to variables by name.

Usage

```
fmapr(
  .df,
  .f,
  ncores = NULL,
  pb = FALSE,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
  .seed = NULL,
  ...
)
```

Arguments

`.df`	A data frame whose rows will be iterated over.
`.f`	A function applied to each row, which receives a named list.
`ncores`	Integer. Number of cores to use for parallel processing. Default is ‘NULL‘ (sequential).
`pb`	Logical. Whether to display a progress bar. Default is ‘FALSE‘.
`.on_error`	How errors thrown by ‘.f‘ are handled: ‘"stop"‘ (default), ‘"pass"‘, or ‘"fill"‘. See [fapply()].
`.fill`	Replacement for failed rows when ‘.on_error = "fill"‘.
`.seed`	Optional single number for reproducible per-row RNG streams.
`...`	Additional arguments passed to ‘.f‘.

Value

A list of results returned by applying ‘.f‘ to each row as a list.

Examples

```
df <- data.frame(name = c("Mister", "Hipster"), age = c(30, 25))

# Create personalized messages
fmapr(df, function(row) paste(row$name, "is", row$age, "years old"))

# Row-wise model formulas
formulas <- data.frame(
  response = c("y1", "y2"),
  predictor = c("x1", "x2"),
  stringsAsFactors = FALSE
)

fmapr(formulas, function(row) {
  reformulate(row$predictor, row$response)
})
```

---

freduce *Functional reduce*

Description

Apply a binary function iteratively over a list or vector, reducing it to a single value or a sequence of intermediate results. This is a wrapper around [Reduce()] that supports optional initial values, right-to-left evaluation, accumulation of intermediate steps, and output simplification.

Usage

```
freduce(
  .x,
  .f,
  .init = NULL,
  .right = FALSE,
  .accumulate = FALSE,
  .simplify = TRUE
)
```

Arguments

.x	A vector or list to reduce.
.f	A binary function to apply. Can be given as a function or quoted (e.g., '\'+\'').
.init	Optional initial value passed to [Reduce()]. If 'NULL', reduction starts from the first two elements.
.right	Logical. If 'TRUE', reduction is performed from right to left.
.accumulate	Logical. If 'TRUE', returns a list of intermediate results (like a scan).
.simplify	Logical. If 'TRUE' and all intermediate results are length 1, the output is simplified to a vector.

Value

A single value (default) or a list/vector of intermediate results if ‘.accumulate = TRUE‘.

Examples

```
freduce(1:5, `+`)                          # => 15
freduce(letters[1:4], paste0)             # => "abcd"
freduce(list(1, 2, 3), `*`)               # => 6
freduce(1:3, `+`, .init = 10)             # => 16
freduce(1:3, paste0, .right = TRUE)       # => "321"
freduce(1:4, `+`, .accumulate = TRUE)     # => c(1, 3, 6, 10)
```

---

frepeat *Repeat an expression or function call multiple times*

Description

Repeats an expression or function evaluation ‘times‘ times. If ‘expr‘ is a function, it is invoked with optional input ‘.x‘ and additional arguments. If ‘expr‘ is a quoted expression, it is evaluated in the parent environment. Supports parallel processing and optional simplification of results.

Usage

```
frepeat(
  .x = NULL,
  times,
  expr,
  simplify = FALSE,
  ncores = NULL,
  pb = FALSE,
  .seed = NULL,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
  ...
)
```

Arguments

`.x`	Optional input passed to ‘expr‘ if ‘expr‘ is a function. Default is ‘NULL‘.
`times`	Integer. Number of repetitions.
`expr`	A function or an unevaluated expression to repeat. If a function, it will be called ‘times‘ times.
`simplify`	Logical. If ‘TRUE‘, attempts to simplify the result using ‘simplify2array()‘. Default is ‘FALSE‘.

ncores	Integer. Number of cores to use for parallel execution. Default is ‘NULL‘ (sequential).
pb	Logical. Whether to display a progress bar. Default is ‘FALSE‘.
.seed	Optional single number. Gives each replicate its own L’Ecuyer-CMRG RNG stream, so a Monte Carlo run is reproducible and independent of ‘ncores‘. The caller’s global RNG state is restored on exit.
.on_error	How errors in ‘expr‘ are handled: ‘"stop"‘ (default), ‘"pass"‘, or ‘"fill"‘. See [fapply()].
.fill	Replacement for failed replicates when ‘.on_error = "fill"‘.
...	Additional arguments passed to the function ‘expr‘ if it is callable.

Value

A list of outputs (or a simplified array if ‘simplify = TRUE‘) from evaluating ‘expr‘ multiple times.

Note

If ‘expr‘ is passed as a function call (not a function or quoted expression), it will be evaluated immediately, not repeated. Use `function(...) \{ ... \}` or `quote(...)` instead.

Examples

```
# Repeat a pure function call
frepeat(times = 3, expr = function() rnorm(1))

# Repeat a function with input `.x`
frepeat(.x = 10, times = 3, expr = function(x) rnorm(1, mean = x))

# Repeat an unevaluated expression (evaluated with `eval()`)
frepeat(times = 2, expr = quote(rnorm(1)))

# Simplify the output to an array
frepeat(times = 3, expr = function() rnorm(1), simplify = TRUE)

# Monte Carlo simulation: estimate coverage of a 95% CI for sample mean
mc_result <- frepeat(
  times = 1000, simplify = TRUE, pb = TRUE, ncores = 1,
  expr = function() {
    sample <- rnorm(30, mean = 0, sd = 1)
    ci <- t.test(sample)$conf.int
    mean(ci[1] <= 0 & 0 <= ci[2])  # true mean inside the interval?
  }
)
mean(mc_result)  # estimated coverage
```

`fwalk` *Walk over a vector or list with side effects*

Description

Applies a function ‘.f‘ to each element of ‘.x‘, typically for its side effects (e.g., printing, writing files). This function is the side-effect-friendly equivalent of ‘fmap()‘. Supports parallel execution and progress bar display.

Usage

```
fwalk(
  .x,
  .f,
  ncores = NULL,
  pb = FALSE,
  .on_error = c("stop", "pass", "fill"),
  .fill = NULL,
  .seed = NULL,
  ...
)
```

Arguments

`.x`	A list or atomic vector of elements to iterate over.
`.f`	A function to apply to each element of ‘.x‘. Should be called primarily for side effects.
`ncores`	Integer. Number of cores to use for parallel processing. Default is ‘NULL‘ (sequential).
`pb`	Logical. Whether to show a progress bar. Default is ‘FALSE‘.
`.on_error`	How errors thrown by ‘.f‘ are handled: ‘"stop"‘ (default), ‘"pass"‘, or ‘"fill"‘. See [fapply()].
`.fill`	Ignored for the return value (kept for signature parity with the other mappers); failed elements simply do not raise when ‘.on_error != "stop"‘.
`.seed`	Optional single number for reproducible per-task RNG streams.
`...`	Additional arguments passed to ‘.f‘.

Value

Invisibly returns ‘.x‘, like ‘purrr::walk()‘.

Examples

```
# Print each element
fwalk(1:3, print)

# Simulate writing files in parallel

fwalk(1:3, function(i) {
  cat(paste("Processing item", i, "\n"))
  Sys.sleep(0.5)
}, ncores = 2, pb = TRUE)
```

Index

fapply, 2
fcompose, 4
fcv, 5
floop, 6
fmap, 7
fmapc, 8
fmapg, 9
fmapn, 10
fmapr, 12
freduce, 13
frepeat, 14
fwalk, 16