Package 'bench'

June 6, 2018

Title High Precision Timing of R Expressions

Version 1.0.1

Description Tools to accurately benchmark and analyze execution times for R expressions.

License GPL-3

URL https://github.com/r-lib/bench

BugReports https://github.com/r-lib/bench/issues

Depends R (>= 3.1)

Imports glue, methods, pillar, profmem, rlang, stats, tibble, utils

Suggests covr, dplyr, forcats, ggbeeswarm, ggplot2, ggridges, mockery, scales, testthat, tidyr (>= 0.8.1), withr

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1.9000

NeedsCompilation yes

Author Jim Hester [aut, cre]

Maintainer Jim Hester <james.f.hester@gmail.com>

Repository CRAN

Date/Publication 2018-06-06 05:03:17 UTC

R topics documented:

as_bench_mark *Coerce to a bench mark object Bench mark objects*

Description

This is typically needed only if you are performing additional manipulations after calling `bench::mark()`.

Usage

```
as_bench_mark(x)
```

Arguments

x	Object to be coerced

autoplot.bench_mark *Autoplot method for bench_mark objects*

Description

Autoplot method for bench_mark objects

Usage

```
## S3 method for class 'bench_mark'
autoplot(object, type = c("beeswarm", "jitter", "ridge",
  "boxplot", "violin"), ...)

## S3 method for class 'bench_mark'
plot(x, ..., type = c("beeswarm", "jitter", "ridge",
  "boxplot", "violin"), y)
```

Arguments

object	A bench_mark object.
type	The type of plot. Plotting geoms used for each type are • beeswarm - `ggbeeswarm::geom_quasirandom()` • jitter - `ggplot2::geom_jitter()` • ridge - `ggridges::geom_density_ridges()` • boxplot - `ggplot2::geom_boxplot()` • violin - `ggplot2::geom_violin()`
...	Additional arguments passed to the plotting geom.
x	A bench_mark object.
y	Ignored, required for compatibility with the `plot()` generic.

Details

This function requires some optional dependencies. ggplot2, tidyr, and depending on the plot type ggbeeswarm, ggridges.

For `type` of `beeswarm` and `jitter` the points are colored by the highest level garbage collection performed during each iteration.

For plots with 2 parameters `ggplot2::facet_grid()` is used to construct a 2d facet. For other numbers of parameters `ggplot2::facet_wrap()` is used instead.

Examples

```
dat <- data.frame(x = runif(10000, 1, 1000), y=runif(10000, 1, 1000))

res <- bench::mark(
  dat[dat$x > 500, ],
  dat[which(dat$x > 500), ],
  subset(dat, x > 500))

if (require(ggplot2)) {

  # Beeswarm plot
  autoplot(res)

  # ridge (joyplot)
  autoplot(res, "ridge")

  # If you want to have the plots ordered by execution time you can do so by
  # ordering factor levels in the expressions.
  if (require(dplyr) && require(forcats)) {

    res %>%
      mutate(expression = forcats::fct_reorder(expression, min, .desc = TRUE)) %>%
      as_bench_mark() %>%
      autoplot("violin")
  }
}
```

`bench_bytes` *Human readable memory sizes*

Description

Construct, manipulate and display vectors of byte sizes. These are numeric vectors, so you can compare them numerically, but they can also be compared to human readable values such as '10MB'.

Usage

```
as_bench_bytes(x)

bench_bytes(x)
```

Arguments

x	A numeric or character vector. Character representations can use shorthand sizes (see examples).

Details

These memory sizes are always assumed to be base 1024, rather than 1000.

Examples

```
bench_bytes("1")
bench_bytes("1K")
bench_bytes("1Kb")
bench_bytes("1KiB")
bench_bytes("1MB")

bench_bytes("1KB") < "1MB"

sum(bench_bytes(c("1MB", "5MB", "500KB")))
```

---

bench_time *Human readable times*

Description

Construct, manipulate and display vectors of elapsed times in seconds. These are numeric vectors, so you can compare them numerically, but they can also be compared to human readable values such as '10ms'.

Usage

```
as_bench_time(x)

bench_time(x)
```

Arguments

x	A numeric or character vector. Character representations can use shorthand sizes (see examples).

Examples

```
bench_time("1")
bench_time("1ns")
bench_time("1us")
bench_time("1ms")
bench_time("1s")

bench_time("100ns") < "1ms"
```

```
sum(bench_time(c("1MB", "5MB", "500KB")))
```

`knit_print.bench_mark` *Custom printing function for bench_mark objects in knitr documents*

Description

By default data columns ('result', 'memory', 'time', 'gc') are omitted when printing in knitr. If you would like to include these columns set the knitr chunk option 'bench.all_columns = TRUE'.

Usage

```
knit_print.bench_mark(x, ..., options)
```

Arguments

`x`	An R object to be printed
`...`	Additional arguments passed to the S3 method. Currently ignored, except two optional arguments `options` and `inline`; see the references below.
`options`	A list of knitr chunk options set in the currently evaluated chunk.

Details

You can set `bench.all_columns = TRUE` to show all columns of the bench mark object.

````
```{r bench.all_columns = TRUE}
bench::mark(
  subset(mtcars, cyl == 3),
  mtcars[mtcars$cyl == 3, ])
```
````

`mark` *Benchmark a series of functions*

Description

Benchmark a list of quoted expressions. Each expression will always run at least twice, once to measure the memory allocation and store results and one or more times to measure timing.

Usage

```
mark(..., min_time = 0.5, iterations = NULL, min_iterations = 1,
  max_iterations = 10000, check = TRUE, filter_gc = TRUE,
  relative = FALSE, env = parent.frame())
```

Arguments

...	Expressions to benchmark, if named the `expression` column will be the name, otherwise it will be the deparsed expression.
min_time	The minimum number of seconds to run each expression, set to `Inf` to always run `max_iterations` times instead.
iterations	If not `NULL`, the default, run each expression for exactly this number of iterations. This overrides both `min_iterations` and `max_iterations`.
min_iterations	Each expression will be evaluated a minimum of `min_iterations` times.
max_iterations	Each expression will be evaluated a maximum of `max_iterations` times.
check	Check if results are consistent. If `TRUE`, checking is done with `all.equal()`, if `FALSE` checking is disabled. If check is a function that function will be called with each pair of results to determine consistency.
filter_gc	If `TRUE` filter iterations that contained at least one garbage collection before summarizing.
relative	If `TRUE` all summaries are computed relative to the minimum execution time rather than absolute time.
env	The environment which to evaluate the expressions

Value

A tibble with the additional summary columns. The following summary columns are computed

- `min` - `bench_time` The minimum execution time.
- `mean` - `bench_time` The arithmetic mean of execution time
- `median` - `bench_time` The sample median of execution time.
- `max` - `bench_time` The maximum execution time.
- `mem_alloc` - `bench_bytes` Total amount of memory allocated by running the expression.
- `itr/sec` - `integer` The estimated number of executions performed per second.
- `n_itr` - `integer` Total number of iterations after filtering garbage collections (if `filter_gc == TRUE`).
- `n_gc` - `integer` Total number of garbage collections performed over all runs.

See Also

`press()` to run benchmarks across a grid of parameters.

Examples

```
dat <- data.frame(x = runif(100, 1, 1000), y=runif(10, 1, 1000))
mark(
  min_time = .1,

  dat[dat$x > 500, ],
  dat[which(dat$x > 500), ],
  subset(dat, x > 500))
```

press *Run setup code and benchmarks across a grid of parameters*

Description

`press()` is used to run `bench::mark()` across a grid of parameters and then *press* the results together.

The parameters you want to set are given as named arguments and a grid of all possible combinations is automatically created.

The code to setup and benchmark is given by one unnamed expression (often delimited by `{`).

If replicates are desired a dummy variable can be used, e.g. `rep = 1:5` for replicates.

Usage

```
press(..., .grid = NULL)
```

Arguments

`...`	If named, parameters to define, if unnamed the expression to run. Only one unnamed expression is permitted.
`.grid`	A pre-build grid of values to use, typically a data.frame or tibble. This is useful if you only want to use a subset of all possible combinations.

Examples

```
# Helper function to create a simple data.frame of the specified dimensions
create_df <- function(rows, cols) {
  as.data.frame(setNames(
    replicate(cols, runif(rows, 1, 1000), simplify = FALSE),
    rep_len(c("x", letters), cols)))
}

# Run 4 data sizes across 3 samples with 2 replicates (24 total benchmarks)
press(
  rows = c(1000, 10000),
  cols = c(10, 100),
  rep = 1:2,
  {
    dat <- create_df(rows, cols)
    bench::mark(
      min_time = .05,
      bracket = dat[dat$x > 500, ],
      which = dat[which(dat$x > 500), ],
      subset = subset(dat, x > 500)
    )
  }
)
```

`summary.bench_mark` *Summarize bench::mark results.*

Description

Summarize bench::mark results.

Usage

```
## S3 method for class 'bench_mark'
summary(object, filter_gc = TRUE, relative = FALSE,
  ...)
```

Arguments

`object`	bench_mark object to summarize.
`filter_gc`	If `TRUE` filter iterations that contained at least one garbage collection before summarizing.
`relative`	If `TRUE` all summaries are computed relative to the minimum execution time rather than absolute time.
`...`	Additional arguments ignored.

Details

If `filter_gc == TRUE` (the default) runs that contain a garbage collection will be removed before summarizing. This is most useful for fast expressions when the majority of runs do not contain a gc. Call `summary(filter_gc = FALSE)` if you would like to compute summaries *with* these times, such as expressions with lots of allocations when all or most runs contain a gc.

Value

A tibble with the additional summary columns. The following summary columns are computed

- `min` - `bench_time` The minimum execution time.
- `mean` - `bench_time` The arithmetic mean of execution time
- `median` - `bench_time` The sample median of execution time.
- `max` - `bench_time` The maximum execution time.
- `mem_alloc` - `bench_bytes` Total amount of memory allocated by running the expression.
- `itr/sec` - `integer` The estimated number of executions performed per second.
- `n_itr` - `integer` Total number of iterations after filtering garbage collections (if `filter_gc == TRUE`).
- `n_gc` - `integer` Total number of garbage collections performed over all runs.

Examples

```
dat <- data.frame(x = runif(10000, 1, 1000), y=runif(10000, 1, 1000))

# `bench::mark()` implicitly calls summary() automatically
results <- bench::mark(
  dat[dat$x > 500, ],
  dat[which(dat$x > 500), ],
  subset(dat, x > 500))

# However you can also do so explicitly to filter gc differently.
summary(results, filter_gc = FALSE)

# Or output relative times
summary(results, relative = TRUE)
```

---

system_time *Return Process CPU and real time that an expression used.*

Description

Return Process CPU and real time that an expression used.

Usage

```
system_time(expr)
```

Arguments

expr	A expression to be timed.

Details

On some systems (such as macOS) the process clock has lower precision than the realtime clock, as a result there may be cases where the process time is larger than the real time for fast expressions.

Value

A bench_time object with two values.

- `process` - The process CPU usage of the expression evaluation.
- `real` - The wallclock time of the expression evaluation.

Examples

```
# This will use ~.5 seconds of real time, but very little process time.
system_time(Sys.sleep(.5))
```

Index