Package ‘bench’

May 4, 2023

Title  High Precision Timing of R Expressions
Version  1.1.3
Description  Tools to accurately benchmark and analyze execution times for
R expressions.
License  MIT + file LICENSE
BugReports  https://github.com/r-lib/bench/issues
Depends  R (>= 3.5.0)
Imports  glue (>= 1.6.2), methods, pillar (>= 1.9.0), profmem (>=
0.6.0), rlang (>= 0.2.0), stats, tibble (>= 3.2.1), utils
Suggests  covr, dplyr, forcats, ggbeeswarm, ggplot2, ggridges,
parallel, scales, testthat (>= 3.1.8), tidyr (>= 0.8.1), vctrs,
withr

Config/testthat/edition  3

Encoding  UTF-8
RoxygenNote  7.2.3

NeedsCompilation  yes
Author  Jim Hester [aut],
Davis Vaughan [aut, cre],
Drew Schmidt [ctb] (read_proc_file implementation),
Posit Software, PBC [cph, fnd]
Maintainer  Davis Vaughan <davis@posit.co>
Repository  CRAN
Date/Publication  2023-05-04 16:50:02 UTC

\

R topics documented:

  as_bench_mark  ................................................................. 2
  as_bench_time  ................................................................. 2
  autoplot.bench_mark  .......................................................... 3
as_bench_time

### Description

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

### Usage

```r
as_bench_mark(x)
```

### Arguments

- `x` Object to be coerced

---

as_bench_time

### 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

```r
as_bench_time(x)
```

### Arguments

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

```r
as_bench_time("1ns")
as_bench_time("")
as_bench_time("1us")
as_bench_time("1ms")
as_bench_time("1s")

as_bench_time("100ns") < "1ms"

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

Description

Autoplot method for bench_mark objects

Usage

```r
autoplot.bench_mark(
  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()`

- `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

```r
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) && require(tidyr) && require(ggbeeswarm)) {
  # 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(as.character(expression), min, .desc = TRUE)) %>%
    as_bench_mark() %>%
    autoplot("violin")
  }
}
```

benchmark

`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)`
**bench_load_average**

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

```r
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")))
```

---

**Description**

Uses OS system APIs to return the load average for the past 1, 5 and 15 minutes.

**Usage**

```r
bench_load_average()
```

---

**bench_memory**

Measure memory that an expression used.

**Description**

Measure memory that an expression used.

**Usage**

```r
bench_memory(expr)
```

**Arguments**

expr A expression to be measured.
bench_process_memory

Value

A tibble with two columns

- The total amount of memory allocated
- The raw memory allocations as parsed by `profmem::readRprofmem()`

Examples

```r
if (capabilities("profmem")) {
  bench_memory(1 + 1:10000)
}
```

Description

The memory reported here will likely differ from that reported by `gc()`, as this includes all memory from the R process, including any child processes and memory allocated outside R’s garbage collector heap.

Usage

`bench_process_memory()`

Details

The OS APIs used are as follows

**Windows:**

- PROCESS_MEMORY_COUNTERS.WorkingSetSize
- PROCESS_MEMORY_COUNTERS.PeakWorkingSetSize

**macOS:**

- task_info(TASK_BASIC_INFO)
- rusage.ru_maxrss

**linux:**

- /proc/pid/status VmSize
- /proc/pid/status VmPeak
**Description**

Measure Process CPU and real time that an expression used.

**Usage**

```
bench_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.

**See Also**

- `bench_memory()` To measure memory allocations for a given expression.

**Examples**

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

Time is expressed as seconds since some arbitrary time in the past; it is not correlated in any way to
the time of day, and thus is not subject to resetting or drifting. The hi-res timer is ideally suited to
performance measurement tasks, where cheap, accurate interval timing is required.

USAGE

hires_time()

EXAMPLES

hires_time()

# R rounds doubles to 7 digits by default, see greater precision by setting
# the digits argument when printing
print(hires_time(), digits = 20)

# Generally used by recording two times and then subtracting them
start <- hires_time()
end <- hires_time()
elapsed <- end - start
elapsed

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
```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,
  memory = capabilities("profmem"),
  filter_gc = TRUE,
  relative = FALSE,
  time_unit = NULL,
  exprs = NULL,
  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 if results are consistent. If TRUE, checking is done with `all.equal()`, if FALSE checking is disabled and results are not stored. If check is a function that function will be called with each pair of results to determine consistency.

If TRUE (the default when R is compiled with memory profiling), track memory allocations using `utils::Rprofmem()`. If FALSE disable memory tracking.

If TRUE remove iterations that contained at least one garbage collection before summarizing. If TRUE but an expression had a garbage collection in every iteration, filtering is disabled, with a warning.

If TRUE all summaries are computed relative to the minimum execution time rather than absolute time.

If NULL the times are reported in a human readable fashion depending on each value. If one of 'ns', 'us', 'ms', 's', 'm', 'h', 'd', 'w' the time units are instead expressed as nanoseconds, microseconds, milliseconds, seconds, hours, minutes, days or weeks respectively.

A list of quoted expressions. If supplied overrides expressions defined in `...`.

The environment which to evaluate the expressions

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

- `expression` - `bench_expr` The deparsed expression that was evaluated (or its name if one was provided).
- `min` - `bench_time` The minimum execution time.
- `median` - `bench_time` The sample median of execution time.
- `itr/sec` - `double` The estimated number of executions performed per second.
- `mem_alloc` - `bench_bytes` Total amount of memory allocated by R while running the expression. Memory allocated outside the R heap, e.g. by `malloc()` or `new` directly is not tracked, take care to avoid misinterpreting the results if running code that may do this.
- `gc/sec` - `double` The number of garbage collections per second.
- `n_itr` - `integer` Total number of iterations after filtering garbage collections (if `filter_gc` == TRUE).
- `n_gc` - `double` Total number of garbage collections performed over all iterations. This is a pseudo-measure of the pressure on the garbage collector, if it varies greatly between to alternatives generally the one with fewer collections will cause fewer allocation in real usage.
- `total_time` - `bench_time` The total time to perform the benchmarks.
- `result` - `list` A list column of the object(s) returned by the evaluated expression(s).
- `memory` - `list` A list column with results from `Rprofmem()`.
- `time` - `list` A list column with `bench_time` vectors for each evaluated expression.
- `gc` - `list` A list column with tibbles containing the level of garbage collection (0-2, columns) for each iteration (rows).

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))

Description

press() is used to run 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-built grid of values to use, typically a data.frame or tibble. This is useful if you only want to benchmark 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)
summary.bench_mark

`bench::mark`

```
min_time = .05,
      bracket = dat[dat$x > 500, ],
         which = dat[which(dat$x > 500), ],
      subset = subset(dat, x > 500)
```
Value

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

- **expression** - `bench_expr` The deparsed expression that was evaluated (or its name if one was provided).
- **min** - `bench_time` The minimum execution time.
- **median** - `bench_time` The sample median of execution time.
- **itr/sec** - `double` The estimated number of executions performed per second.
- **mem_alloc** - `bench_bytes` Total amount of memory allocated by R while running the expression. Memory allocated outside the R heap, e.g. by `malloc()` or `new` directly is not tracked, take care to avoid misinterpreting the results if running code that may do this.
- **gc/sec** - `double` The number of garbage collections per second.
- **n_itr** - `integer` Total number of iterations after filtering garbage collections (if `filter_gc == TRUE`).
- **n_gc** - `double` Total number of garbage collections performed over all iterations. This is a pseudo-measure of the pressure on the garbage collector, if it varies greatly between to alternatives generally the one with fewer collections will cause fewer allocation in real usage.
- **total_time** - `bench_time` The total time to perform the benchmarks.
- **result** - `list` A list column of the object(s) returned by the evaluated expression(s).
- **memory** - `list` A list column with results from `Rprofmem()`.
- **time** - `list` A list column of `bench_time` vectors for each evaluated expression.
- **gc** - `list` A list column with tibbles containing the level of garbage collection (0-2, columns) for each iteration (rows).

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)
workout a group of expressions individually

**Description**

Given an block of expressions in {} `workout()` individually times each expression in the group. `workout_expressions()` is a lower level function most useful when reading lists of calls from a file.

**Usage**

```r
workout(expr, description = NULL)
workout_expressions(exprs, env = parent.frame(), description = NULL)
```

**Arguments**

- `expr` one or more expressions to workout, use {} to pass multiple expressions.
- `description` A name to label each expression, if not supplied the deparsed expression will be used.
- `exprs` A list of calls to measure.
- `env` The environment in which the expressions should be evaluated.

**Examples**

```r
workout(
  {
    x <- 1:1000
    evens <- x %% 2 == 0
    y <- x[evens]
    length(y)
    length(which(evens))
    sum(evens)
  }
)
```

# The equivalent to the above, reading the code from a file
```r
workout_expressions(as.list(parse(system.file("examples/exprs.R", package = "bench")))
```

# The equivalent to the above, reading the code from a file
```r
workout_expressions(as.list(parse(system.file("examples/exprs.R", package = "bench")))
```
Index

all.equal(), 10
as_bench_bytes (bench_bytes), 4
as_bench_mark, 2
as_bench_time, 2
autoplot.bench_mark, 3
bench_bytes, 4
bench_load_average, 5
bench_mark, 12
bench_mark (mark), 9
bench_memory, 5
bench_memory (), 7
bench_process_memory, 6
bench_time, 7
data.frame, 11
ggbeeswarm, 4
ggbeeswarm::geom_quasirandom (), 3
ggplot2, 4
ggplot2::geom_boxplot (), 3
ggplot2::geom_jitter (), 3
ggplot2::geom_violin (), 3
ggridges, 4
ggridges::geom_density_ridges (), 3
hires_time, 8
knit_print.bench_mark, 8
mark, 9, 12
mark (), 2, 11
plot.bench_mark (autoplot.bench_mark), 3
press, 11
press (), 10
profmem::readRprofmem (), 6
Rprofmem (), 10, 13
summary.bench_mark, 12
system_time (bench_time), 7
tibble, 10, 11, 13
tidy, 4
utils::Rprofmem (), 10
workout, 14
workout (), 14
workout_expressions (workout), 14
workout_expressions (), 14