Package ‘wrswoR’

October 12, 2022

Type Package
Title Weighted Random Sampling without Replacement
Version 1.1.1
Date 2020-07-26
Description A collection of implementations of classical and novel algorithms for weighted sampling without replacement.
License GPL-3
URL http://krlmlr.github.io/wrswoR
BugReports https://github.com/krlmlr/wrswoR/issues
Depends R (>= 3.0.2)
Imports logging (>= 0.4-13), Rcpp
Suggests BatchExperiments, BiocManager, dplyr, ggplot2, import,kimisc
       (>= 0.2-4), knitr, metap, microbenchmark, markdown, roxygen2, rtables (>= 0.1), sampling, testthat, tidyr, tikzDevice (>= 0.9-1)
LinkingTo Rcpp (>= 0.11.5)
Encoding UTF-8
LazyData true
RoxygenNote 7.1.1.9000
URLNote https://github.com/krlmlr/wrswoR
NeedsCompilation yes
Author Kirill Müller [aut, cre]
Maintainer Kirill Müller <krlmlr+r@mailbox.org>
Repository CRAN
Date/Publication 2020-07-26 18:20:02 UTC

R topics documented:

wrswoR-package ................................................................. 2
sample_int_crank ............................................................. 2
**Description**

R’s default sampling without replacement using `base::sample.int()` seems to require quadratic run time, e.g., when using weights drawn from a uniform distribution. For large sample sizes, this is too slow. This package contains several alternative implementations.

**Details**

Implementations are adapted from [https://stackoverflow.com/q/15113650/946850](https://stackoverflow.com/q/15113650/946850).

**Author(s)**

Kirill Müller

**References**


**Examples**

```r
sample_int_rej(100, 50, 1:100)
```

**Description**

These functions implement weighted sampling without replacement using various algorithms, i.e., they take a sample of the specified size from the elements of `1:n` without replacement, using the weights defined by `prob`. The call `sample_int_*(n, size, prob)` is equivalent to `sample.int(n, size, replace = FALSE, prob)`. (The results will most probably be different for the same random seed, but the returned samples are distributed identically for both calls.) Except for `sample_int_R()` (which has quadratic complexity as of this writing), all functions have complexity $O(n \log n)$ or better and often run faster than R’s implementation, especially when `n` and `size` are large.
**Usage**

```r
sample_int_crank(n, size, prob)
sample_int_ccrank(n, size, prob)
sample_int_expj(n, size, prob)
sample_int_expjs(n, size, prob)
sample_int_R(n, size, prob)
sample_int_rank(n, size, prob)
sample_int_rej(n, size, prob)
```

**Arguments**

- `n`: a positive number, the number of items to choose from. See ‘Details.’
- `size`: a non-negative integer giving the number of items to choose.
- `prob`: a vector of probability weights for obtaining the elements of the vector being sampled.

**Details**

- `sample_int_R()` is a simple wrapper for `base::sample.int()`.
- `sample_int_expj()` and `sample_int_expjs()` implement one-pass random sampling with a reservoir with exponential jumps (Efraimidis and Spirakis, 2006, Algorithm A-ExpJ). Both functions are implemented in Rcpp; `_expj()` uses log-transformed keys, `_expjs()` implements the algorithm in the paper verbatim (at the cost of numerical stability).
- `sample_int_rank()`, `sample_int_crank()` and `sample_int_ccrank()` implement one-pass random sampling (Efraimidis and Spirakis, 2006, Algorithm A). The first function is implemented purely in R, the other two are optimized Rcpp implementations (`_crank()` uses R vectors internally, while `_ccrank()` uses std::vector; surprisingly, `_crank()` seems to be faster on most inputs). It can be shown that the order statistic of $U^{1/w_i}$ has the same distribution as random sampling without replacement ($U =$ uniform$(0,1)$ distribution). To increase numerical stability, $\log(U)/w_i$ is computed instead; the log transform does not change the order statistic.
- `sample_int_rej()` uses repeated weighted sampling with replacement and a variant of rejection sampling. It is implemented purely in R. This function simulates weighted sampling without replacement using somewhat more draws with replacement, and then discarding duplicate values (rejection sampling). If too few items are sampled, the routine calls itself recursively on a (hopefully) much smaller problem. See also [http://stats.stackexchange.com/q/20590/6432](http://stats.stackexchange.com/q/20590/6432).

**Value**

An integer vector of length `size` with elements from 1:n.
Author(s)
Dinre (for ._rank()), Kirill Müller (for all other functions)

References

https://stackoverflow.com/q/15113650/946850


See Also

base::sample.int()

Examples

```R
# Base R implementation
s <- sample_int_R(2000, 1000, runif(2000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_R(6, 3, p),
                      n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A, Rcpp version using std::vector
s <- sample_int_ccrank(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_ccrank(6, 3, p),
                      n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A, Rcpp version using R vectors
s <- sample_int_crank(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_crank(6, 3, p),
                      n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A-ExpJ (with log-transformed keys)
s <- sample_int_expj(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
```

sample_int_crank

```r
tbl <- table(replicate(sample_int_expj(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A-ExpJ (paper version)
s <- sample_int_expjs(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_expjs(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A
s <- sample_int_rank(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_rank(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Rejection sampling
s <- sample_int_rej(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_rej(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)
```
Index

* package
  wrswoR-package, 2

base::sample.int(), 2–4

sample_int(sample_int_crank), 2
sample_int_crank(sample_int_crank), 2
sample_int_crank, 2
sample_int_expj(sample_int_crank), 2
sample_int_expjs(sample_int_crank), 2
sample_int_R(sample_int_crank), 2
sample_int_rank(sample_int_crank), 2
sample_int_rej(sample_int_crank), 2

wrswoR (wrswoR-package), 2
wrswoR-package, 2