# Package: InPlotSampling (via r-universe)

May 26, 2026

**Type** Package

**Title** Easing the Application of Ranked Set Sampling in Practice

**Version** 0.1.0

**Date** 2021-02-09

**Description** The InPlotSampling package provides a way for researchers to easily implement Ranked Set Sampling in practice. Ranked Set Sampling was originally described by McIntyre (1952) (reprinted in 2005) <doi:10.1198/000313005X54180>. This package takes work by Omer and Kravchuk (2021) <https://doi.org/10.1007/s13253-021-00439-1> and enables easy use of the methods.

**License** MIT + file LICENSE

**URL** https://aagi-aus.github.io/InPlotSampling/

**BugReports** https://github.com/AAGI-AUS/InPlotSampling/issues

**Depends** R (>= 3.5.0)

**Imports** Rcpp, ggplot2, SDaA, FNN

**Suggests** covr, testthat, parallel

**LinkingTo** Rcpp

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Repository** https://aagi-aus.r-universe.dev

**Date/Publication** 2026-03-27 06:43:03 UTC

**RemoteUrl** https://github.com/AAGI-AUS/InPlotSampling

**RemoteRef** HEAD

**RemoteSha** 5a444efee2ec3af20769a732b67d3e569735ac87

# Contents



---

`bootstrap_sample` *Generate boostrap sample on the provided population.*

---

## Description

Generate boostrap sample on the provided population.

## Usage

```
bootstrap_sample(pop, n, n_bootstraps)
```

## Arguments

| | |
|---|---|
| `pop` | Population data |
| `n` | Sample sizes (SBS sample size, PPS sample size). |
| `n_bootstraps` | Number of bootstrap samples. |

## Value

A summary data frame of the estimator.

## coombe2019 *Coombe Vineyard Data from 2019 Season.*

### Description

Measurements taken on the Coombe Research Vineyard, University of Adelaide Waite Campus after the 2019 season.

### Usage

```
coombe2019
```

### Format

A data frame with 352 rows and 11 variables:

**vine_id** Identifier for the individual vines (1-352)

**rootstock** The rootstock that the vine is growing on (8 levels)

**row** The row in the vineyard (10-20)

**panel** The panel of the vines. A pair of vines is grouped into a panel (1-16)

**trunk_circ_18** The trunk circumference of the vine in 2018, measured in cm at watering height (~20 cm above the ground).

**trunk_circ_19** The trunk circumference of the vine in 2019, measured in cm at watering height (~20 cm above the ground).

**count_shoots** Check this?

**non_count_shoots** Check this?

**total_shoots** Sum of `count_shoot` and `non_count_shoot`.

**pruning_weight** Weight of the material removed during pruning in Kg. Check this?

**cordon_length** The length of the cordon in cm

## emergence_ranks *Ranks for Seed Emergence.*

### Description

Contains the ranks given by 5 rankers of the number of seeds emerged in a sample of 15 plots.

### Usage

```
emergence_ranks
```

### Format

A data frame with 6 variables: `seed_emergence`, `ranker1`, `ranker2`, `ranker3`, `ranker4` and `ranker5`.

## InPlotSampling *InPlotSampling: Easing the Application of Ranked Set Sampling in Practice*

### Description

The InPlotSampling package provides a way for researchers to easily implement Ranked Set Sampling in practice. Ranked Set Sampling was originally described by McIntyre (1952) (reprinted in 2005) doi:10.1198/000313005X54180. This package takes work by Omer and Kravchuk (2021) https://doi.org/10.1007/s13253-021-00439-1 and enables easy use of the methods.

### Author(s)

**Maintainer**: Sam Rogers <sam.rogers@adelaide.edu.au>

Authors:

- Omer Ozturk <omer@stat.osu.edu> (ORCID)
- Olena Kravchuk <olena.kravchuk@adelaide.edu.au> (ORCID) [data contributor]
- Peter Kasprzak <peter.kasprzak@adelaide.edu.au>

### See Also

Useful links:

- https://aagi-aus.github.io/InPlotSampling/
- Report bugs at https://github.com/AAGI-AUS/InPlotSampling/issues

## jps_estimate *Computes the estimator for JPS data*

### Description

Computes the estimator for JPS data

### Usage

```
jps_estimate(data, set_size, replace = TRUE, model_based, N, alpha)
```

### Arguments

| | |
|---|---|
| data | The data to use for estimation. |
| set_size | Set size for each ranking group. |
| replace | Logical (default TRUE). Sample with replacement? |
| model_based | An inference mode:<br>• FALSE: design based inference<br>• TRUE: model based inference using super population model |
| N | The population size. |
| alpha | The significance level. |

### Value

A data.frame with the point estimates provided by JPS estimators along with standard error and confidence intervals.

---

## jps_sample *Generate JPS sampling on the provided population.*

---

### Description

Generate JPS sampling on the provided population.

### Usage

```
jps_sample(pop, n, H, tau, K, replace = FALSE, with_index = FALSE)
```

### Arguments

| | |
|---|---|
| pop | Population that will be sampled. |
| n | Sample size. |
| H | Set size for each ranking group. |
| tau | A parameter which controls ranking quality. |
| K | Number of rankers. |
| replace | A boolean which specifies whether to sample with replacement or not. |
| with_index | A boolean which specifies whether to return the index of the sampled population. |

### Value

A matrix with ranks from each ranker.

## Examples

```
set.seed(112)
population_size <- 600
# the number of samples to be ranked in each set
H <- 3

with_replacement <- FALSE
sigma <- 4
mu <- 10
n_rankers <- 3
# sample size
n <- 10

rhos <- rep(0.75, n_rankers)
taus <- sigma * sqrt(1 / rhos^2 - 1)

population <- qnorm((1:population_size) / (population_size + 1), mu, sigma)
jps_sample(population, n, H, taus, n_rankers, with_replacement)
#>               Y R1 R2 R3
#>  [1,]  6.384461  1  2  2
#>  [2,]  1.485141  1  1  1
#>  [3,] 13.640711  2  3  2
#>  [4,] 15.809136  3  3  2
#>  [5,]  6.769463  2  2  1
#>  [6,] 14.355524  3  3  3
#>  [7,] 10.729740  2  1  3
#>  [8,]  6.152453  1  1  1
#>  [9,]  8.701285  2  1  2
#> [10,] 13.323884  3  3  3
```

---

`population` *Seed Emergence Population.*

---

## Description

The entire population of actual and estimated seed emergence.

## Usage

```
population
```

## Format

A data frame with 2640 rows of 2 variables: `actual_seed_emergence` and `estimated_seed_emergence` giving the actual and estimated number of seeds emerged.

`rss_jps_estimate` *Estimate means from RSS or JPS sample.*

## Description

Estimate means from RSS or JPS sample.

## Usage

```
rss_jps_estimate(
  data,
  set_size,
  method,
  confidence = 0.95,
  replace = TRUE,
  model_based = FALSE,
  pop_size = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame of `JPS` or `RSS` rankings. |
| `set_size` | The set size of the ranks. |
| `method` | A method used to sample:<br>• `"JPS"`: Judgment-post stratified sampling<br>• `"RSS"`: Ranked set sampling |
| `confidence` | The confidence level to use. |
| `replace` | Logical (default `TRUE`). Sample with replacement? |
| `model_based` | An inference mode:<br>• `FALSE`: design based inference<br>• `TRUE`: model based inference using super population model |
| `pop_size` | The population size. Must be provided if<br>• sampling without replacement, or<br>• `model_based` is `TRUE`. |

## Value

A `data.frame` with the point estimates provided by different types of estimators along with standard error and confidence intervals.

## Examples

```
# JPS estimator
set.seed(112)
population_size <- 600
# the number of samples to be ranked in each set
H <- 3

with_replacement <- FALSE
sigma <- 4
mu <- 10
n_rankers <- 3
# sample size
n <- 30

rhos <- rep(0.75, n_rankers)
taus <- sigma * sqrt(1 / rhos^2 - 1)
population <- qnorm((1:population_size) / (population_size + 1), mu, sigma)

data <- InPlotSampling::jps_sample(population, n, H, taus, n_rankers, with_replacement)
data <- data[order(data[, 2]), ]

rss_jps_estimate(
  data,
  set_size = H,
  method = "JPS",
  confidence = 0.80,
  replace = with_replacement,
  model_based = FALSE,
  pop_size = population_size
)
#>          Estimator Estimate Standard Error 80% Confidence intervals
#> 1       UnWeighted    9.570          0.526               8.88,10.26
#> 2      Sd.Weighted    9.595          0.569             8.849,10.341
#> 3 Aggregate Weight    9.542          0.500             8.887,10.198
#> 4     JPS Estimate    9.502          0.650             8.651,10.354
#> 5     SRS estimate    9.793          0.783             8.766,10.821
#> 6          Minimum    9.542          0.500             8.887,10.198

# RSS estimator
set.seed(112)
population_size <- 600
# the number of samples to be ranked in each set
H <- 3

with_replacement <- FALSE
sigma <- 4
mu <- 10
n_rankers <- 3
# sample size
n <- 30

population <- qnorm((1:population_size) / (population_size + 1), mu, sigma)
```

```
rho <- 0.75
tau <- sigma * sqrt(1 / rho^2 - 1)
x <- population + tau * rnorm(population_size, 0, 1)

population <- cbind(population, x)
data <- InPlotSampling::rss_sample(population, n, H, n_rankers, with_replacement)
data <- data[order(data[, 2]), ]

rss_estimates <- rss_jps_estimate(
  data,
  set_size = H,
  method = "RSS",
  confidence = 0.80,
  replace = with_replacement,
  model_based = FALSE,
  pop_size = population_size
)

print(rss_estimates)
#>             Estimator point.est St.error 80% Confidence Interval
#> 1               RSS-1     9.153    0.766            8.148,10.158
#> 2  Aggregate Weighted     9.064    0.652             8.209,9.919
```

---

`rss_sample` *Generate ranked set sampling (RSS) on the population provided.*

---

## Description

Generate ranked set sampling (RSS) on the population provided.

## Usage

```
rss_sample(pop, n, H, K, replace = FALSE)
```

## Arguments

| | |
|---|---|
| `pop` | Population that will be sampled with an auxiliary parameter in the second column. |
| `n` | Sample size. |
| `H` | Set size for each ranking group. |
| `K` | Number of rankers. |
| `replace` | A boolean which specifies whether to sample with replacement or not. |

## Value

A matrix with ranks from each ranker.

## Examples

```
set.seed(112)
population_size <- 600
# the number of samples to be ranked in each set
H <- 3

replace <- FALSE
sigma <- 4
mu <- 10
n_rankers <- 3
# sample size
n <- 9

population <- qnorm((1:population_size) / (population_size + 1), mu, sigma)
rho <- 0.75
tau <- sigma * sqrt(1 / rho^2 - 1)
x <- population + tau * rnorm(population_size, 0, 1)

population <- cbind(population, x)
rss_sample(population, n, H, n_rankers, replace)
#>            [,1] [,2] [,3] [,4]
#>  [1,]  8.910625    1    3    1
#>  [2,] 12.317145    2    2    1
#>  [3,] 11.619746    3    3    2
#>  [4,]  8.307549    1    2    1
#>  [5,]  5.089992    2    2    3
#>  [6,]  7.233575    3    3    3
#>  [7,] 11.601654    1    2    2
#>  [8,]  9.134107    2    1    1
#>  [9,] 12.960431    3    3    1
```

---

`sbs_pps_estimate` *Compute an estimator for SBS PPS sampled data.*

---

## Description

Compute an estimator for SBS PPS sampled data.

## Usage

```
sbs_pps_estimate(
  population,
  n,
  y,
  sample_matrix,
  n_bootstraps = 100,
  alpha = 0.05,
  n_cores = getOption("n_cores", 1)
)
```

## Arguments

| | |
|---|---|
| population | Population data frame to be sampled with 4 columns.<br>1. Halton numbers<br>2. X1-coordinate of population unit<br>3. X2-coordinate of population unit<br>4. Size measurements of population units |
| n | Sample sizes (SBS sample size, PPS sample size). |
| y | Sample response values. |
| sample_matrix | Sample data frame to be sampled with 6 columns.<br>1. Halton numbers<br>2. X1-coordinate of population unit<br>3. X2-coordinate of population unit<br>4. Size measurement of population unit<br>5. Weight<br>6. Inclusion probability |
| n_bootstraps | Number of bootstrap samples. |
| alpha | The significance level. |
| n_cores | The number of cores to be used for computational tasks (specify 0 for max). This can also be set by calling options, e.g., options(n_cores = 2). |

## Value

A summary data frame of the estimator.

## Examples

```
set.seed(112)

# SBS sample size, PPS sample size
sample_sizes <- c(5, 5)

n_population <- 233
k <- 0:(n_population - 1)
x1 <- sample(1:13, n_population, replace = TRUE) / 13
x2 <- sample(1:8, n_population, replace = TRUE) / 8
y <- (x1 + x2) * runif(n = n_population, min = 1, max = 2) + 1
measured_sizes <- y * runif(n = n_population, min = 0, max = 4)

population <- matrix(cbind(k, x1, x2, measured_sizes), ncol = 4)
sample_result <- sbs_pps_sample(population, sample_sizes)

# estimate the population mean and construct a confidence interval
df_sample <- sample_result$sample
sample_id <- df_sample[, 1]
y_sample <- y[sample_id]
```

```
sbs_pps_estimates <- sbs_pps_estimate(
  population, sample_sizes, y_sample, df_sample,
  n_bootstrap = 100, alpha = 0.05
)
print(sbs_pps_estimates)
#>   n1 n2 Estimate  St.error 95% Confidence intervals
#> 1  5  5    2.849 0.1760682              2.451,3.247
```

---

`sbs_pps_heatmap` *Generate a heat map of SBS PPS sample of the provided population.*

---

## Description

Generate a heat map of SBS PPS sample of the provided population.

## Usage

```
sbs_pps_heatmap(pop, sbs_indices, pps_indices)
```

## Arguments

| | |
|---|---|
| `pop` | Population data frame to be sampled with 5 columns.<br>1. Halton numbers<br>2. X1-coordinate of population unit<br>3. X2-coordinate of population unit<br>4. Size measurements of population units<br>5. Inclusion probabilities |
| `sbs_indices` | Indices of SBS sample. |
| `pps_indices` | Indices of PPS sample. |

## Value

Heat map of the sample.

`sbs_pps_sample` *Generate probability proportional to size (PPS) and spatially balanced sampling on the population provided.*

## Description

Generate probability proportional to size (PPS) and spatially balanced sampling on the population provided.

## Usage

```
sbs_pps_sample(population, n, n_cores = getOption("n_cores", 1))
```

## Arguments

| | |
|---|---|
| `population` | Population data frame to be sampled with 4 columns.<br>1. Halton numbers<br>2. X1-coordinate of population unit<br>3. X2-coordinate of population unit<br>4. Size measurements of population units |
| `n` | Sample sizes (SBS sample size, PPS sample size). |
| `n_cores` | The number of cores to be used for computational tasks (specify 0 for max). This can also be set by calling `options`, e.g., `options(n_cores = 2)`. |

## Value

A named list of:

- heatmap: heat map of the sample
- sample: SBS PPS sample of the population

## Examples

```
set.seed(112)

# SBS sample size, PPS sample size
sample_sizes <- c(5, 5)

n_population <- 233
k <- 0:(n_population - 1)
x1 <- sample(1:13, n_population, replace = TRUE) / 13
x2 <- sample(1:8, n_population, replace = TRUE) / 8
y <- (x1 + x2) * runif(n = n_population, min = 1, max = 2) + 1
measured_sizes <- y * runif(n = n_population, min = 0, max = 4)

population <- matrix(cbind(k, x1, x2, measured_sizes), ncol = 4)
sample_result <- sbs_pps_sample(population, sample_sizes)
```

```
print(sample_result$sample)
#>    sbs_pps_indices        x1    x2       size      weight inclusion_probability
#> 1               87 0.4615385 0.625  0.4665423 0.000000000            0.02319163
#> 2               88 0.1538462 0.625  1.7389902 0.000000000            0.02790409
#> 3               89 0.8461538 0.625  7.0815547 0.000000000            0.04749104
#> 4               90 0.6923077 0.750  9.5428032 0.000000000            0.05640733
#> 5               91 0.2307692 0.750  5.1375136 0.000000000            0.04039996
#> 6              173 0.1538462 0.500  6.6400168 0.005024130            0.04588620
#> 7               26 0.6153846 0.500  4.3146186 0.003264631            0.03738898
#> 8              232 0.8461538 0.750 12.0057856 0.009084108            0.06526583
#> 9              171 0.6153846 0.750  6.9029083 0.005223046            0.04684225
#> 10              29 0.8461538 0.375  4.6324720 0.003505133            0.03855377
```

---

`single_bootstrap` *Generate a single boostrap sample on the provided population.*

---

## Description

Generate a single boostrap sample on the provided population.

## Usage

```
single_bootstrap(pop, n)
```

## Arguments

| | |
|---|---|
| `pop` | Population data |
| `n` | Sample sizes (SBS sample size, PPS sample size). |

## Value

A summary data frame of the estimator.

---

`two_stage_cluster_sample`
*Generate two-stage cluster sampling on the population provided.*

---

## Description

Generate two-stage cluster sampling on the population provided.

## Usage

```
two_stage_cluster_sample(
  pop,
  sampling_strategies,
  n,
  H,
  replace,
  ni,
  Hi,
  replace_i
)
```

## Arguments

| | |
|---|---|
| pop | Population that will be sampled with these ordered columns:<br>1. Parent id: an index to denotes the parent of the record<br>2. Parent auxiliary parameter: an auxiliary parameter for ranking parents<br>3. Child auxiliary parameter: an auxiliary parameter for ranking children |
| sampling_strategies | (first stage sampling strategy, second stage sampling strategy), e.g., `c('srs', 'jps')`.<br>• `'srs'`: simple random sampling without replacement<br>• `'jps'`: JPS sampling |
| n | Number of samples in the first stage. |
| H | Set size for each ranking group in the first stage. |
| replace | A boolean which specifies whether to sample with replacement or not in the first stage (applicable only for JPS sampling). |
| ni | Number(s) of samples in the second stage. Can be a single number or a vector of `n` numbers. |
| Hi | Set size for each ranking group in the second stage. Can be a single number or a vector of `n` numbers. |
| replace_i | A boolean which specifies whether to sample with replacement or not in the second stage (applicable only for JPS sampling). |

## Value

A matrix with ranks from each ranker.

## Examples

```
set.seed(112)
parent_size <- 300
child_size <- 50
# the number of samples to be ranked in each set
H <- 3
```

```
sampling_strategies <- c("jps", "jps")
replace <- FALSE
mu <- 10
sigma <- 4
n <- 4

parent_indices <- rep(1:parent_size, child_size)
parent_aux <- abs(qnorm(1:parent_size / (parent_size + 1), mu, sigma) + 5 * rnorm(parent_size, 0, 1))
child_aux <- abs(parent_aux + 10 * rnorm(parent_size * child_size, 0, 1))

population <- cbind(parent_indices, rep(parent_aux, child_size), child_aux)
two_stage_cluster_sample(population, sampling_strategies, n, H, replace, 6, 3, FALSE)
#>       parent_id parent_rank child_id  child_aux child_rank
#>  [1,]       201           1     7101  2.2349453          1
#>  [2,]       201           1    12801  9.7175545          3
#>  [3,]       201           1     6501  7.9207230          1
#>  [4,]       201           1     9801  5.7644835          2
#>  [5,]       201           1    10701 13.8089335          3
#>  [6,]       201           1     3501  0.3598331          1
#>  [7,]       254           2     8654 17.3059292          3
#>  [8,]       254           2    11354 15.0837335          2
#>  [9,]       254           2     9254  6.0103919          2
#> [10,]       254           2     2954 12.7011502          2
#> [11,]       254           2    14954  5.1158133          2
#> [12,]       254           2    13754  5.8931551          1
#> [13,]        74           1     8474  4.3393349          1
#> [14,]        74           1     9674 15.0512523          2
#> [15,]        74           1     6674 12.9022479          3
#> [16,]        74           1      674  2.9209174          2
#> [17,]        74           1     7274  7.2500468          3
#> [18,]        74           1     6374  7.0925954          1
#> [19,]       223           3     9223 28.4694257          3
#> [20,]       223           3      223  4.4001977          1
#> [21,]       223           3     9823 22.8676415          3
#> [22,]       223           3    11923 26.4531048          3
#> [23,]       223           3      823 20.8714211          2
#> [24,]       223           3     9523  8.1783058          1
```

# Index