

# Package ‘OPWeight’

November 28, 2024

**Type** Package

**Title** Optimal p-value weighting with independent information

**Version** 1.29.0

**Date** 2017-02-26

**Description** This package perform weighted-pvalue based multiple hypothesis test and provides corresponding information such as ranking probability, weight, significant tests, etc . To conduct this testing procedure, the testing method apply a probabilistic relationship between the test rank and the corresponding test effect size.

**Depends** R (>= 3.4.0),

**License** Artistic-2.0

**LazyData** true

**Imports** graphics, qvalue, MASS, tibble, stats,

**Suggests** airway, BiocStyle, cowplot, DESeq2, devtools, ggplot2, gridExtra, knitr, Matrix, rmarkdown, scales, testthat

**VignetteBuilder** knitr

**biocViews** ImmunoOncology, BiomedicalInformatics, MultipleComparison, Regression, RNASeq, SNP

**RoxygenNote** 6.0.1

**URL** <https://github.com/mshasan/OPWeight>

**Bugreports** <https://github.com/mshasan/OPWeight/issues>

**git\_url** <https://git.bioconductor.org/packages/OPWeight>

**git\_branch** devel

**git\_last\_commit** 1f5f472

**git\_last\_commit\_date** 2024-10-29

**Repository** Bioconductor 3.21

**Date/Publication** 2024-11-27

**Author** Mohamad Hasan [aut, cre],  
Paul Schliekelman [aut]

**Maintainer** Mohamad Hasan <shakilmohamad7@gmail.com>

## Contents

|  |    |
|--|----|
| opw . . . . .                          | 2  |
| prob_rank_givenEffect . . . . .        | 5  |
| prob_rank_givenEffect_approx . . . . . | 6  |
| prob_rank_givenEffect_exact . . . . .  | 8  |
| prob_rank_givenEffect_simu . . . . .   | 9  |
| weight_binary . . . . .                | 11 |
| weight_by_delta . . . . .              | 12 |
| weight_continuous . . . . .            | 13 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>15</b> |
|--------------|-----------|

---

|     |   |
|-----|---|
| opw | <i>Perform Optimal Pvalue Weighting</i> |
|-----|---|

---

### Description

A function to perform weighted pvalue multiple hypothesis test. This function compute the probabilities of the ranks of the filter statistics given the effect sizes, and consequently the weights if neither the weights nor the probabilities are given. Then provides the number of rejected null hypothesis and the list of the rejected pvalues as well as the corresponding filter statistics.

### Usage

```
opw(pvalue, filter, weight = NULL, ranksProb = NULL,
    mean_filterEffect = NULL, mean_testEffect = NULL,
    effectType = c("continuous", "binary"), alpha = 0.05, nrep = 10000,
    tail = 1L, delInterval = 0.001, method = c("BH", "BON"), ...)
```

### Arguments

|                   |  |
|-------------------|--|
| pvalue            | Numeric vector of pvalues of the test statistics   |
| filter            | Numeric vector of filter statistics  |
| weight            | An optional numeric weight vector not required   |
| ranksProb         | An optional numeric vector of the ranks probability of the filters given the mean effect   |
| mean_filterEffect | Numeric, value of the mean filter effect of the true alternatives  |
| mean_testEffect   | Numeric, value of the mean test effect of the true alternatives  |
| effectType        | Character ("continuous" or "binary"), type of effect sizes   |
| alpha             | Numeric, significance level of the hypothesis test   |
| nrep              | Integer, number of replications for importance sampling, default value is 10,000, can be increased to obtain smoother probability curves |

|             |  |
|-------------|--|
| tail        | Integer (1 or 2), right-tailed or two-tailed hypothesis test. default is right-tailed test.  |
| delInterval | Numeric, interval between the delta values of a sequence. Note that, delta is a LaGrange multiplier, necessary to normalize the weight |
| method      | Character ("BH" or "BON"), type of methods is used to obtain the results; Benjemini-Hochberg or Bonferroni                             |
| ...         | Arguments passed to internal functions   |

## Details

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i > 0,$$

then the `mean_testEffect` and `mean_filterEffect` should be mean of the test and filter effect sizes, respectively. This is called hypothesis testing for the continuous effect sizes.

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i = \epsilon,$$

then `mean_testEffect` and `mean_filterEffect` should be median or any discrete value of the test and filter effect sizes. This is called hypothesis testing for the Binary effect sizes, where `epsilon` refers to a fixed value.

The main goal of the function is to compute the probabilities of the ranks from the pvalues and the filter statistics, consequently the weights. Although `weights` `ranksProb` are optional, `opw` has the options so that one can compute the probabilities and the weights externally if necessary (see examples).

Internally, `opw` function compute the `ranksProb` and consequently the weights, then uses the pvalues to make conclusions about hypotheses. Therefore, if `ranksProb` is given then `mean_filterEffect` and `mean_testEffect` are redundant, and should not be provided to the function. Although `ranksProb` is not required to the function, One can compute `ranksProb` by using the function [prob\\_rank\\_givenEffect](#).

The function internally compute `mean_filterEffect` and `mean_testEffect` from a simple linear regression with box-cox transformation between the test and filter statistics, where the filters are regressed on the test statistics. Thus, filters need to be positive to apply `boxcox` from the R library MASS. Then the estimated `mean_filterEffect` and `mean_testEffect` are used to obtain the `ranksProb` and the weights. Thus, in order to apply the function properly, it is crucial to understand the uses `mean_filterEffect` and `mean_testEffect`. If `mean_filterEffect` and `mean_testEffect` are not provided then the test statistics computed from the pvalues will be used to compute the relationship between the filter statistics and the test statistics.

If one of the mean effects `mean_filterEffect` and `mean_testEffect` are not provided then the missing mean effect will be computed internally.

**Value**

totalTests Integer, total number of hypothesis tests evaluated  
 nullProp Numeric, estimated proportion of the true null hypothesis  
 ranksProb Numeric vector of ranks probability given the mean filter effect,  $p(\text{rank} | \text{ey} = \text{mean\_filterEffect})$   
 weight Numeric vector of normalized weight  
 rejections Integer, total number of rejections  
 rejections\_list data frame, list of rejected p-values and the corresponding filter statistics and the adjusted p-values if method = "BH" used.

**Author(s)**

Mohamad S. Hasan, shakilmohamad7@gmail.com

**See Also**

[prob\\_rank\\_givenEffect](#) [weight\\_binary](#) [weight\\_continuous](#) [qvalue](#) [dnorm](#)

**Examples**

```
# generate pvalues and filter statistics
m = 1000
set.seed(3)
filters = runif(m, min = 0, max = 2.5)           # filter statistics
H = rbinom(m, size = 1, prob = 0.1)             # hypothesis true or false
tests = rnorm(m, mean = H * filters)            # Z-score
pvals = 1 - pnorm(tests)                       # pvalue

# general use
results <- opw(pvalue = pvals, filter = filters, effectType = "continuous",
              method = "BH")

# supply the mean effects for both the filters and the tests externally
mod <- lm(log(filters) ~ tests)
et = mean(tests)
ey = mod$coef[[1]] + mod$coef[[2]]*et
results2 <- opw(pvalue = pvals, filter = filters,
              mean_filterEffect = ey, mean_testEffect = et, tail = 2,
              effectType = "continuous", method = "BH")

# supply the rank probabilities externally
library(qvalue)
ranks <- 1:m
nullProp = qvalue(p = pvals, pi0.method = "bootstrap")$pi0
m0 = ceiling(nullProp*m)
m1 = m - m0
probs <- sapply(ranks, prob_rank_givenEffect, et = ey, ey = ey,
              nrep = 10000, m0 = m0, m1 = m1)
results3 <- opw(pvalue = pvals, filter = filters, ranksProb = probs,
              effectType = "continuous", tail = 2, method = "BH")
```

```
# supply weight externally
wgt <- weight_continuous(alpha = .05, et = et, m = m, ranksProb = probs)
results4 <- opw(pvalue = pvals, filter = filters, weight = wgt,
               effectType = "continuous", alpha = .05, method = "BH")
```

---

prob\_rank\_givenEffect *Probability of rank of test given effect size*

---

### Description

Compute the probability of rank of a test being higher than any other tests given the effect size from external information.

### Usage

```
prob_rank_givenEffect(k, et, ey, nrep = 10000, m0, m1)
```

### Arguments

|      |  |
|------|--|
| k    | Integer, rank of a test                                      |
| et   | Numeric, effect of the targeted test for importance sampling |
| ey   | Numeric, mean filter effect from the external information    |
| nrep | Integer, number of replications for importance sampling      |
| m0   | Integer, number of true null hypothesis                      |
| m1   | Integer, number of true alternative hypothesis               |

### Details

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i > 0,$$

then ey should be mean of the filter effect sizes, This is called hypothesis testing for the continuous effect sizes.

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i = \epsilon,$$

then ey should be median or any discrete value of the filter effect sizes. This is called hypothesis testing for the Binary effect sizes.

If monitor = TRUE then a window will open to see the progress of the computation. It is useful for a large number of tests

m1 and m0 can be estimated using qvalue from a bioconductor package qvalue.

**Value**

prob Numeric, probability of the rank of a test

**Author(s)**

Mohamad S. Hasan, shakilmohamad7@gmail.com

**See Also**

[dnorm](#) [pnorm](#) [rnorm](#) [qvalue](#)

**Examples**

```
# compute the probability of the rank of a test being third if all tests are
# from the true null
prob <- prob_rank_givenEffect(k = 3, et = 0, ey = 0, nrep = 10000,
                             m0 = 50, m1 = 50)

# compute the probabilities of the ranks of a test being rank 1 to 100 if the
# targeted test effect is 2 and the overall mean filter effect is 1.
ranks <- 1:100
prob <- sapply(ranks, prob_rank_givenEffect, et = 2, ey = 1, nrep = 10000,
              m0 = 50, m1 = 50)

# plot
plot(ranks,prob)
```

---

prob\_rank\_givenEffect\_approx

*Probability of rank of test given effect size by normal approximation*

---

**Description**

A normal approximation to compute the probability of rank of a test being higher than any other test given the effect size from external information.

**Usage**

```
prob_rank_givenEffect_approx(k, et, ey, nrep = 10000, m0, m1,
                             effectType = c("binary", "continuous"))
```

**Arguments**

|    |  |
|----|--|
| k  | Integer, rank of a test                                      |
| et | Numeric, effect of the targeted test for importance sampling |
| ey | Numeric, mean/median filter effect from external information |

|            |  |
|------------|--|
| nrep       | Integer, number of replications for importance sampling    |
| m0         | Integer, number of true null hypothesis                    |
| m1         | Integer, number of true alternative hypothesis             |
| effectType | Character ("continuous" or "binary"), type of effect sizes |

### Details

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i > 0,$$

then ey should be mean of the filter effect sizes, This is called hypothesis testing for the continuous effect sizes.

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i = \epsilon,$$

then ey should be median or any discrete value of the filter effect sizes. This is called hypothesis testing for the Binary effect sizes.

m1 and m0 can be estimated using qvalue from a bioconductor package qvalue.

### Value

prob Numeric, probability of the rank of a test

### Author(s)

Mohamad S. Hasan, shakilmohamad7@gmail.com

### See Also

[dnorm](#) [pnorm](#) [rnorm](#) [qvalue](#)

### Examples

```
# compute the probability of the rank of a test being third if all tests are
# from the true null
prob <- prob_rank_givenEffect(k = 3, et = 0, ey = 0, nrep = 10000,
                             m0 = 50, m1 = 50)

# compute the probabilities of the ranks of a test being rank 1 to 100 if the
# targeted test effect is 2 and the overall mean filter effect is 1.
ranks <- 1:100
prob <- sapply(ranks, prob_rank_givenEffect, et = 2, ey = 1, nrep = 10000,
              m0 = 50, m1 = 50)

# plot
plot(ranks,prob)
```

---

 prob\_rank\_givenEffect\_exact

*Probability of rank of test given effect size by exact method*


---

### Description

An exact method to compute the probability of rank of a test being higher than any other test given the effect size from external information.

### Usage

```
prob_rank_givenEffect_exact(k, et, ey, nrep = 10000, m0, m1,
  effectType = c("binary", "continuous"))
```

### Arguments

|            |  |
|------------|--|
| k          | Integer, rank of a test                                      |
| et         | Numeric, effect of the targeted test for importance sampling |
| ey         | Numeric, mean/median filter effect from external information |
| nrep       | Integer, number of replications for importance sampling      |
| m0         | Integer, number of true null hypothesis                      |
| m1         | Integer, number of true alternative hypothesis               |
| effectType | Character ("continuous" or "binary"), type of effect sizes   |

### Details

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i > 0,$$

then ey should be mean of the filter effect sizes, This is called hypothesis testing for the continuous effect sizes.

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i = \epsilon,$$

then ey should be median or any discrete value of the filter effect sizes. This is called hypothesis testing for the Binary effect sizes.

m1 and m0 can be estimated using qvalue from a bioconductor package qvalue.

### Value

prob Numeric, probability of the rank of a test

### Author(s)

Mohamad S. Hasan, shakilmohamad7@gmail.com



**See Also**

[dnorm](#) [pnorm](#) [rnorm](#) [qvalue](#)

**Examples**

```
# compute the probability of the rank of a test being third if all tests are
# from the true null
prob <- prob_rank_givenEffect_exact(k=3, et=0, ey=0, nrep=10000, m0=50, m1=50,
                                   effectType= "continuous")

# compute the probabilities of the ranks of a test being rank 1 to 100 if the
# targeted test effect is 2 and the overall mean filter effect is 1.
ranks <- 1:100
prob <- sapply(ranks, prob_rank_givenEffect, et = 2, ey = 1, nrep = 10000,
              m0 = 50, m1 = 50)

# plot
plot(ranks, prob)
```

---

prob\_rank\_givenEffect\_simu

*Probability of rank of test given effect size by simulations*

---

**Description**

A simulation approach to compute the probability of rank of a test being higher than any other test given the effect size from the external information.

**Usage**

```
prob_rank_givenEffect_simu(s, ey, e.one, m0, m1, effectType = c("binary",
  "continuous"))
```

**Arguments**

|            |   |
|------------|---|
| s          | number of samples of test statistics composed of null and alternative tests |
| ey         | Numeric, filter test effect from the external information                   |
| e.one      | Numeric, one test effect that will vary across all tests                    |
| m0         | Integer, number of true null hypothesis                                     |
| m1         | Integer, number of true alternative hypothesis                              |
| effectType | Character ("continuous" or "binary"), type of effect sizes                  |

**Details**

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i > 0,$$

then they should be mean of the filter effect sizes, This is called hypothesis testing for the continuous effect sizes.

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i = \epsilon,$$

then they should be median or any discrete value of the filter effect sizes. This is called hypothesis testing for the Binary effect sizes.

This is a simulation approach to compute the probability of the rank,  $P(\text{rank} \mid \text{effect} = \epsilon)$  to verify the actual  $P(\text{rank} \mid \text{effect} = \epsilon)$ . Suppose, we have a vector of  $m = m_1 + m_0$  observations, where the first  $m_1$  observations are from the true alternative and second  $m_0$  are from the true null models. If we pick two tests one from the first position and the other from the  $(m_0 + 1)$ -th position, then we would expect that the first observation's rank is greater than  $m_0$ , and  $(m_1 + 1)$ -th observation's rank is less than or equal to  $m_1$ . However, this is not always true, especially when the effect size of the test statistics is low, but the above scenario becomes obvious as the effect size increases.  $m_1$  and  $m_0$  can be estimated using `qvalue` from a bioconductor package `qvalue`.

**Value**

`r0` Integer, rank of the null test statistic

`r1` Integer, rank of the alternative test statistic

**Author(s)**

Mohamad S. Hasan, shakilmohamad7@gmail.com

**See Also**

[runif](#) [rnorm](#) [qvalue](#)

**Examples**

```
# total number of sample generated (use sample size at least 1,000,000)
sampleSize = 10000
m0 = 50
m1 = 50
m = m0 + m1

# compute rank of the tests
rank <- sapply(1:sampleSize, prob_rank_givenEffect_simu, ey = 1, e.one = 1,
              m0 = m0, m1 = m1, effectType = "continuous")

# rank may generate missing value because of the large effect size,
# therefore, to make a matplot one needs vector of equal size. This procedure
# will replace the missing value to make the equal sized vectors
```

```

# probability of the rank of a null test
prob0 <- rep(NA, m)
prob0_x <- tapply(rank[1,], rank[1,], length)/sampleSize
prob0[as.numeric(names(prob0_x))] <- as.vector(prob0_x)

# probability of the rank of an alternative test
prob1 <- rep(NA, m)
prob1_x <- tapply(rank[2,], rank[2,], length)/sampleSize
prob1[as.numeric(names(prob1_x))] <- as.vector(prob1_x)

# plot
matplot(1:m, cbind(prob0, prob1), type = "l")

```

---

weight\_binary

*Weight for the Binary effect sizes*


---

### Description

Compute weight from the probability of the rank given the effect size for the binary effect size

### Usage

```
weight_binary(alpha, et, m, m1, tail = 1L, delInterval = 0.001, ranksProb)
```

### Arguments

|             |  |
|-------------|--|
| alpha       | Numeric, significance level of the hypothesis test   |
| et          | Numeric, mean effect size of the test statistics   |
| m           | Integer, total number of hypothesis test   |
| m1          | Integer, number of true alternative hypothesis   |
| tail        | Integer (1 or 2), right-tailed or two-tailed hypothesis test. default is right-tailed test.  |
| delInterval | Numeric, interval between the delta values of a sequence. Note that, delta is a LaGrange multiplier, necessary to normalize the weight |
| ranksProb   | Numeric vector of the ranks probability of the tests given the effect size   |

### Details

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i = \epsilon,$$

then et and ey should be median or any discrete value of the test and filter effect sizes, respectively. This is called hypothesis testing for the Binary effect sizes. m1 can be estimated using qvalue from a bioconductor package qvalue.

**Value**

weight Numeric vector of normalized weight of the tests for the binary case

**Author(s)**

Mohamad S. Hasan, shakilmohamad7@gmail.com

**See Also**

[prob\\_rank\\_givenEffect](#) [weight\\_continuous](#) [qvalue](#)

**Examples**

```
# compute the probabilities of the ranks of a test being rank 1 to 100 if the
# targeted test effect is 2 and the overall mean filter effect is 1.
ranks <- 1:100
prob2 <- sapply(ranks, prob_rank_givenEffect, et = 2, ey = 1, nrep = 10000,
               m0 = 50, m1 = 50)

# plot the prooabbility
plot(ranks, prob2)

# compute weight for the binary case
weight_bin <- weight_binary(alpha = .05, et = 1, m = 100, m1 = 50, tail=1,
                           delInterval = .0001, ranksProb = prob2)

# plot the weight
plot(ranks, weight_bin)
```

---

weight\_by\_delta

*Find sum of weights for the LaGrange multiplier*

---

**Description**

Compute sum of weights for a given value of the LaGrange multiplier

**Usage**

```
weight_by_delta(delta, alpha = 0.05, et, m, m1, tail = 1L, ranksProb,
                effectType = c("continuous", "binary"))
```

**Arguments**

|       |  |
|-------|--|
| delta | Numeric value of the LagRange multiplier           |
| alpha | Numeric, significance level of the hypothesis test |
| et    | Numeric, mean effect size of the test statistics   |
| m     | Integer, total number of hypothesis test           |

|            |   |
|------------|---|
| m1         | Integer, number of true alternative tests   |
| tail       | Integer (1 or 2), right-tailed or two-tailed hypothesis test. default is right-tailed test. |
| ranksProb  | Numeric vector of the ranks probability of the filter statistics given the effect size      |
| effectType | Character ("continuous" or "binary"), type of effect sizes                                  |

### Details

To obtain the normalized weight, and to make sure that the sum of the weights is equal to the number of tests and the weights are positive, an optimal value of the LaGrange multiplier delta needed. This function will compute the weights for a given value of the LaGrange multiplier and provide the sum of the weights in return.

### Value

sumWeight\_per\_delta sum of weights per delta value

### Author(s)

Mohamad S. Hasan, shakilmohamad7@gmail.com

### Examples

```
# generate a sequence of delta
delta <- seq(0, 1, .0001)

# compute probability fiven effect
filters = runif(100, min = 0, max = 2.5)
probs <- dnorm(filters, mean = 0, sd = 1)

# compute the sum of weights for each delta
weightSum_by_delta <- sapply(delta, weight_by_delta, m = 100, m1 = 50, et = 2,
                             ranksProb = probs, effectType = "continuous")
```

---

weight\_continuous      *Weight for the continuous effect sizes*

---

### Description

Compute weight from the probability of the rank given the effect size for the continuous effect size

### Usage

```
weight_continuous(alpha, et, m, tail = 1L, delInterval = 0.001, ranksProb)
```

**Arguments**

|             |  |
|-------------|--|
| alpha       | Numeric, significance level of the hypothesis test   |
| et          | Numeric, mean effect size of the test statistics   |
| m           | Integer, total number of hypothesis test   |
| tail        | Integer (1 or 2), right-tailed or two-tailed hypothesis test. default is right-tailed test.  |
| delInterval | Numeric, interval between the delta values of a sequence. Note that, delta is a LaGrange multiplier, necessary to normalize the weight |
| ranksProb   | Numeric vector of ranks probability of the tests given the effect size   |

**Details**

If one wants to test

$$H_0 : \epsilon_i = 0 \text{ vs. } H_a : \epsilon_i > 0,$$

then et and ey should be mean value of the test and filter effect sizes, respectively. This is called hypothesis testing for the continuous effect sizes.

**Value**

weight Numeric vector of normalized weight of the tests for the continuous case

**Author(s)**

Mohamad S. Hasan, shakilmohamad7@gmail.com

**See Also**

[prob\\_rank\\_givenEffect](#) [weight\\_binary](#)

**Examples**

```
# compute the probabilities of the ranks of a test being rank 1 to 100 if the
# targeted test effect is 2 and the overall mean filter effect is 1.
ranks <- 1:100
prob2 <- sapply(ranks, prob_rank_givenEffect, et = 2, ey = 1, nrep = 10000,
               m0 = 50, m1 = 50)

# plot the probability
plot(ranks, prob2)

# compute weight for the continuous case
weight_cont <- weight_continuous(alpha = .05, et = 1, m = 100, tail = 1,
                                delInterval = .0001, ranksProb = prob2)

# plot the weight
plot(ranks, weight_cont)
```

# Index

`dnorm`, [4](#), [6](#), [7](#), [9](#)

`opw`, [2](#)

`pnorm`, [6](#), [7](#), [9](#)

`prob_rank_givenEffect`, [3](#), [4](#), [5](#), [12](#), [14](#)

`prob_rank_givenEffect_approx`, [6](#)

`prob_rank_givenEffect_exact`, [8](#)

`prob_rank_givenEffect_simu`, [9](#)

`qvalue`, [4](#), [6](#), [7](#), [9](#), [10](#), [12](#)

`rnorm`, [6](#), [7](#), [9](#), [10](#)

`runif`, [10](#)

`weight_binary`, [4](#), [11](#), [14](#)

`weight_by_delta`, [12](#)

`weight_continuous`, [4](#), [12](#), [13](#)