

# Package ‘permGS’

June 22, 2017

**Title** Permutational Group Sequential Test for Time-to-Event Data

**Version** 0.2.4

**Date** 2017-06-21

**Maintainer** Matthias Brueckner <matthias.brueckner@posteo.de>

**Description** Permutational group-sequential tests for time-to-event data based on the log-rank test statistic. Supports exact permutation test when the censoring distributions are equal in the treatment and the control group and approximate imputation-permutation methods when the censoring distributions are different.

**Depends** survival

**License** GPL-3 | file LICENSE

**Encoding** UTF-8

**LazyData** true

**Imports** stats, coin, utils

**Suggests** testthat

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** Matthias Brueckner [aut, cre],  
Martin Posch [aut],  
Franz Koenig [aut]

**Repository** CRAN

**Date/Publication** 2017-06-22 11:22:46 UTC

## R topics documented:

createPermGS . . . . .	2
exactLR . . . . .	3
imputeHeinze . . . . .	4
imputeIPT . . . . .	4
imputeIPZ . . . . .	5
nextStage . . . . .	6
parseFormula . . . . .	7

permGS . . . . .	7
permHeinze . . . . .	8
permIPT . . . . .	9
permIPZ . . . . .	10
permLR . . . . .	11
permuteHeinze . . . . .	12
permuteIPT . . . . .	13
permuteIPZ . . . . .	13
sampleFromCondKM . . . . .	14
sampleFromKM . . . . .	15
shuffleBlock . . . . .	15
summary.permGS . . . . .	16
<b>Index</b>	<b>17</b>

---

createPermGS	<i>createPermGS</i>
--------------	---------------------

---

## Description

Create permGS object representing a permutational group-sequential trial.

## Usage

```
createPermGS(B = 1000, restricted = TRUE, method = "IPZ", pool = TRUE,
  type = c("logrank", "Gehan-Breslow", "Tarone-Ware", "Prentice",
    "Prentice-Marek", "Andersen-Borgan-Gill-Keiding", "Fleming-Harrington",
    "Self"), imputeData = NULL, permuteData = NULL)
```

## Arguments

B	number of random permutations
restricted	if TRUE only permute within strata
method	imputation/permutation method IPZ, IPT, Heinze or none (default: IPZ)
pool	if TRUE impute event times from Kaplan-Meier estimator calculated from pooled data
type	logrank weights to be used with coin::logrank_trafo
imputeData	user-supplied imputation function (ignored if method is given)
permuteData	user-supplied permutation function (ignore if method is given)

## Value

object of class permGS

**Examples**

```
## standard permutation test (no imputation, free permutations)
x <- createPermGS(1000, FALSE, "none")
summary(x)
## imputation using IPT method, restricted permutations
y <- createPermGS(1000, TRUE, "IPT")
summary(y)
```

---

<code>exactLR</code>	<i>exactLR</i>
----------------------	----------------

---

**Description**

One-sided exact / approximate permutation and asymptotic log-rank test

**Usage**

```
exactLR(B, formula, data = parent.frame(), type = "exact")
```

**Arguments**

<code>B</code>	number of random permutations (only used if <code>type="approximate"</code> )
<code>formula</code>	a formula object, as used by <a href="#">coxph</a> , left hand side must be a 'Surv' object, right hand side may only consist of a single term (treatment indicator)
<code>data</code>	data.frame or list containing the variables in "formula", by default "formula" is evaluated in the parent frame
<code>type</code>	if <code>type="exact"</code> performs complete enumeration of all permutations, if <code>type="approximate"</code> draw random permutations, if <code>type="asymptotic"</code> perform asymptotic log-rank test

**Details**

This function performs a standard exact or approximate permutation test which is only valid under the extended null hypothesis of equal survival AND censoring distributions.

**Value**

A list containing the exact or approximate permutation p-value and the observed test statistic

**Examples**

```
T <- rexp(20)
C <- rexp(20)
data <- data.frame(time=pmin(T, C), status=(T<=C), trt=rbinom(20, 1, 0.5))

# Approximate permutation test using 1000 random permutations
x <- exactLR(1000, Surv(time, status) ~ trt, data, "approximate")
```

```

print(paste("Approximate permutation p-value:", x$p))

# Exact permutation test
y <- exactLR(0, Surv(time, status) ~ trt, data, "exact")
print(paste("Exact permutation p-value:", y$p))

```

---

imputeHeinze	<i>imputeHeinze</i>
--------------	---------------------

---

### Description

Impute data according to Heinze et al. method. Output is supposed to be passed to `permute.heinze`

### Usage

```
imputeHeinze(data, pool = TRUE)
```

### Arguments

<code>data</code>	matrix as returned by <code>as.matrix(generateData(param))</code>
<code>pool</code>	if TRUE impute events times from pooled Kaplan-Meier estimator (default: TRUE)

### Value

list containing Kaplan-Meier estimators of censoring and survival distributions and the original data

### References

Heinze, G., Gnant, M. and Schemper, M. Exact Log-Rank Tests for Unequal Follow-Up. *Biometrics*, 59(4), December 2003.

---

imputeIPT	<i>imputeIPT</i>
-----------	------------------

---

### Description

Impute data according to IPT method. Output is supposed to be passed to `permute.IPT`

### Usage

```
imputeIPT(data, pool = TRUE)
```

**Arguments**

data	matrix as returned by <code>as.matrix(generateData(param))</code>
pool	if TRUE impute events times from pooled Kaplan-Meier estimator (default: TRUE)

**Value**

matrix containing imputed survival and censoring times (columns 1 and 2), and original treatment indicator (column 3)

**References**

Wang, R., Lagakos, S.-W. and Gray, R.-J. Testing and interval estimation for two-sample survival comparisons with small sample sizes and unequal censoring. *Biostatistics*, 11(4), 676–692, January 2010.

---

imputeIPZ

---

*imputeIPZ*


---

**Description**

Impute data according to IPZ method. Output is supposed to be passed to `permute.IPZ`

**Usage**

```
imputeIPZ(data, pool = TRUE)
```

**Arguments**

data	matrix as returned by <code>as.matrix(generateData(param))</code>
pool	if TRUE impute events times from pooled Kaplan-Meier estimator (default: TRUE)

**Value**

original data with 4 new columns (V1 and V2) containing the imputed observations

**References**

Wang, R., Lagakos, S.-W. and Gray, R.-J. Testing and interval estimation for two-sample survival comparisons with small sample sizes and unequal censoring. *Biostatistics*, 11(4), 676–692, January 2010.

---

nextStage

*nextStage*


---

## Description

Imputation permutation group-sequential log-rank test. Random permutations of a block are reused in all later stages. This automatically results in blockwise permutations.

## Usage

```
nextStage(pgs.obj, alpha, formula, data = parent.frame())
```

## Arguments

pgs.obj	permGS object as returned by <a href="#">createPermGS</a>
alpha	alpha at current stage
formula	a formula object, as used by <a href="#">coxph</a> , left hand side must be a 'Surv' object, right hand side must only consist of a factor (treatment indicator) and optionally a special strata() term identifying the permutation strata
data	a data.frame or list containing the variables in "formula", by default "formula" is evaluated in the parent frame

## Value

An updated permGS object.

## Examples

```
## Two-stage design with one-sided O'Brien-Fleming boundaries using IPZ method
x <- createPermGS(1000, TRUE, "IPZ")

t1 <- 9 ## calendar time of interim analysis
t2 <- 18 ## calendar time of final analysis

T <- rexp(100) ## event times
R <- runif(100, 0, 12) ## recruitment times
Z <- rbinom(100, 1, 0.5) ## treatment assignment
C <- rexp(100) ## drop-out times

## Stage 1 data
data.t1 <- data.frame(time=pmin(T, C, max(0, (t1-R))), status=(T<=pmin(C, t1-R)), trt=Z)
data.t1 <- data.t1[R <= t1,]

## Stage 2 data
data.t2 <- data.frame(time=pmin(T, C, max(0, (t2-R))), status=(T<=pmin(C, t2-R)), trt=Z)
data.t2 <- data.t2[R <= t2,]
x <- nextStage(x, 0.00153, Surv(time, status) ~ trt, data.t1)
summary(x)
```

```

if(!x$results$reject[1]) {
  data.t2$strata <- rep.int(c(1,2), c(nrow(data.t1), nrow(data.t2)-nrow(data.t1)))
  x <- nextStage(x, 0.025, Surv(time, status) ~ trt + strata(strata), data.t2)
  summary(x)
}

```

---

parseFormula

*Parse formula of survival model*


---

### Description

Parse formula of survival model

### Usage

```
parseFormula(formula, data = parent.frame())
```

### Arguments

formula	formula object
data	data.frame (optional)

### Value

data.frame containing the parsed variables

---

permGS

*permGS*


---

### Description

This package implements permutational group-sequential tests for time-to-event data based on (weighted) log-rank test statistics. It supports exact permutation test when the censoring distributions are equal in the treatment and the control group and the approximate imputation-permutation methods of Heinze et al. (2003) and Wang et al. (2010) and when the censoring distributions are different. Permutations can be stratified, i.e. only patients within the same stratum are treated as exchangeable. Rejection boundaries are monotone and finite even when only a random subset of all permutations is used. One- and Two-sided testing possible.

### Author(s)

Matthias Brueckner <m.bruckner@lancaster.ac.uk>, Franz Koenig <Franz.Koenig@meduniwien.ac.at>, Martin Posch <martin.posch@meduniwien.ac.at>

## References

- Brueckner, M., Koenig, F. and Posch, M. Group-sequential permutation tests for time-to-event data. Heinze, G., Gnant, M. and Schemper, M. Exact Log-Rank Tests for Unequal Follow-Up. *Biometrics*, 59(4), December 2003.
- Wang, R., Lagakos, S.-W. and Gray, R.-J. Testing and interval estimation for two-sample survival comparisons with small sample sizes and unequal censoring. *Biostatistics*, 11(4), 676–692, January 2010.
- Kelly, P., Zhou, Y., Whitehead, N. J., Stallard, N. and Bowman, C. Sequentially testing for a gene–drug interaction in a genomewide analysis. *Statistics in Medicine*, 27(11), 2022–2034, May 2008.

## Examples

```
## IPZ method based on logrank test with 1000 restricted random permutations
x <- createPermGS(1000, TRUE, "IPZ", type="logrank")

T <- rexp(100) ## event times
R <- runif(100, 0, 12) ## recruitment times
Z <- rbinom(100, 1, 0.5) ## treatment assignment
C <- rexp(100) ## drop-out times

## two-stage design
t1 <- 9 ## calendar time of interim analysis
t2 <- 18 ## calendar time of final analysis

## Stage 1
data.t1 <- data.frame(time=pmin(T, C, max(0, (t1-R))), status=(T<=pmin(C, t1-R)), trt=Z)
data.t1 <- data.t1[R <= t1,]
x <- nextStage(x, 0.00153, Surv(time, status) ~ trt, data.t1)
summary(x)

if(!x$results$reject[1]) { ## Stage 2
  data.t2 <- data.frame(time=pmin(T, C, max(0, (t2-R))), status=(T<=pmin(C, t2-R)), trt=Z)
  data.t2 <- data.t2[R <= t2,]
  data.t2$strata <- rep.int(c(1,2), c(nrow(data.t1), nrow(data.t2)-nrow(data.t1)))
  x <- nextStage(x, alpha=0.025, Surv(time, status) ~ trt + strata(strata), data.t2)
  summary(x)
}
```

---

permHeinze

---

*Convenience function which calls createPermGS and nextStage to perform fixed sample size permutation test with Heinze method*


---

## Description

Convenience function which calls createPermGS and nextStage to perform fixed sample size permutation test with Heinze method

**Usage**

```
permHeinze(formula, data, B = 1000, alpha = 0.05, pool = TRUE,
  type = c("logrank", "Gehan-Breslow", "Tarone-Ware", "Prentice",
    "Prentice-Marek", "Andersen-Borgan-Gill-Keiding", "Fleming-Harrington",
    "Self"))
```

**Arguments**

formula	a formula object, as used by <code>coxph</code> , left hand side must be a 'Surv' object, right hand side must only consist of a factor (treatment indicator) and optionally a special <code>strata()</code> term identifying the permutation strata
data	a data.frame or list containing the variables in "formula", by default "formula" is evaluated in the parent frame
B	number of random permutations (default: 1000)
alpha	significance level (default: 0.05)
pool	if TRUE impute event times from Kaplan-Meier estimator calculated from pooled data
type	logrank weights to be used with <code>coin::logrank_trafo</code>

**Value**

An object of class `permGS`

**Examples**

```
T <- rexp(30) ## event times
Z <- rbinom(30, 1, 0.5) ## treatment assignment
C <- rexp(30) ## drop-out times
data <- data.frame(time=pmin(T,C), status=T<=C, Z=Z)
x <- permHeinze(Surv(time, status) ~ Z, data)
summary(x)
```

---

permIPT	<i>Convenience function which calls <code>createPermGS</code> and <code>nextStage</code> to perform fixed sample size permutation test with IPT method</i>
---------	--

---

**Description**

Convenience function which calls `createPermGS` and `nextStage` to perform fixed sample size permutation test with IPT method

**Usage**

```
permIPT(formula, data, B = 1000, alpha = 0.05, pool = TRUE,
  type = c("logrank", "Gehan-Breslow", "Tarone-Ware", "Prentice",
    "Prentice-Marek", "Andersen-Borgan-Gill-Keiding", "Fleming-Harrington",
    "Self"))
```

**Arguments**

formula	a formula object, as used by <code>coxph</code> , left hand side must be a 'Surv' object, right hand side must only consist of a factor (treatment indicator) and optionally a special <code>strata()</code> term identifying the permutation strata
data	a data.frame or list containing the variables in "formula", by default "formula" is evaluated in the parent frame
B	number of random permutations (default: 1000)
alpha	significance level (default: 0.05)
pool	if TRUE impute event times from Kaplan-Meier estimator calculated from pooled data
type	logrank weights to be used with <code>coin::logrank_trafo</code>

**Value**

An object of class `permGS`

**Examples**

```
T <- rexp(30) ## event times
Z <- rbinom(30, 1, 0.5) ## treatment assignment
C <- rexp(30) ## drop-out times
data <- data.frame(time=pmin(T,C), status=T<=C, Z=Z)
x <- permIPT(Surv(time, status) ~ Z, data)
summary(x)
```

---

permIPZ	<i>Convenience function which calls <code>createPermGS</code> and <code>nextStage</code> to perform fixed sample size permutation test with IPZ method</i>
---------	--

---

**Description**

Convenience function which calls `createPermGS` and `nextStage` to perform fixed sample size permutation test with IPZ method

**Usage**

```
permIPZ(formula, data, B = 1000, alpha = 0.05, pool = TRUE,
  type = c("logrank", "Gehan-Breslow", "Tarone-Ware", "Prentice",
    "Prentice-Marek", "Andersen-Borgan-Gill-Keiding", "Fleming-Harrington",
    "Self"))
```

**Arguments**

formula	a formula object, as used by <code>coxph</code> , left hand side must be a 'Surv' object, right hand side must only consist of a factor (treatment indicator) and optionally a special <code>strata()</code> term identifying the permutation strata
data	a data.frame or list containing the variables in "formula", by default "formula" is evaluated in the parent frame
B	number of random permutations (default: 1000)
alpha	significance level (default: 0.05)
pool	if TRUE impute event times from Kaplan-Meier estimator calculated from pooled data
type	logrank weights to be used with <code>coin::logrank_trafo</code>

**Value**

An object of class `permGS`

**Examples**

```
T <- rexp(30) ## event times
Z <- rbinom(30, 1, 0.5) ## treatment assignment
C <- rexp(30) ## drop-out times
data <- data.frame(time=pmin(T,C), status=T<=C, Z=Z)
x <- permIPZ(Surv(time, status) ~ Z, data)
summary(x)
```

---

permLR	<i>Convenience function which calls <code>createPermGS</code> and <code>nextStage</code> to perform fixed sample size permutation test without imputation</i>
--------	---

---

**Description**

Convenience function which calls `createPermGS` and `nextStage` to perform fixed sample size permutation test without imputation

**Usage**

```
permLR(formula, data, B = 1000, alpha = 0.05, pool = TRUE,
  type = c("logrank", "Gehan-Breslow", "Tarone-Ware", "Prentice",
    "Prentice-Marek", "Andersen-Borgan-Gill-Keiding", "Fleming-Harrington",
    "Self"))
```

**Arguments**

formula	a formula object, as used by <code>coxph</code> , left hand side must be a 'Surv' object, right hand side must only consist of a factor (treatment indicator) and optionally a special <code>strata()</code> term identifying the permutation strata
data	a data.frame or list containing the variables in "formula", by default "formula" is evaluated in the parent frame
B	number of random permutations (default: 1000)
alpha	significance level (default: 0.05)
pool	if TRUE impute event times from Kaplan-Meier estimator calculated from pooled data
type	logrank weights to be used with <code>coin::logrank_trafo</code>

**Value**

An object of class `permGS`

**Examples**

```
## Two-sided permutation test
T <- rexp(100) ## event times
Z <- rbinom(100, 1, 0.5) ## treatment assignment
C <- rexp(100) ## drop-out times
data <- data.frame(time=pmin(T,C), status=T<=C, Z=Z)
x <- permLR(Surv(time, status) ~ Z, data, alpha=c(0.025, 0.025))
summary(x)
```

---

permuteHeinze

---

*permuteHeinze*


---

**Description**

Perform single imputation and permutation step

**Usage**

```
permuteHeinze(imp, pp, index = TRUE)
```

**Arguments**

imp	list as returned by <code>impute.heinze</code>
pp	vector of permuted indices
index	not used

**Value**

matrix with time, status, trt columns

**References**

Heinze, G., Gnant, M. and Schemper, M. Exact Log-Rank Tests for Unequal Follow-Up. *Biometrics*, 59(4), December 2003.

---

permuteIPT

*permuteIPT*


---

**Description**

Permute survival times after imputation (IPT)

**Usage**

```
permuteIPT(data, pp, index = TRUE)
```

**Arguments**

data	matrix as returned by impute.IPT
pp	vector of permuted indices
index	not used

**Value**

matrix with time, status, trt columns

**References**

Wang, R., Lagakos, S.~W. and Gray, R.~J. Testing and interval estimation for two-sample survival comparisons with small sample sizes and unequal censoring. *Biostatistics*, 11(4), 676–692, January 2010.

---

permuteIPZ

*permuteIPZ*


---

**Description**

Permute treatment assignment after imputation (IPZ)

**Usage**

```
permuteIPZ(data, pZ, index = FALSE)
```

**Arguments**

data	matrix as returned by impute.IPT
pZ	vector of permuted indices if index is TRUE, else binary vector of treatment assignments
index	indicates if pZ is a vector of indices or a binary vector of treatment assignments

**Value**

matrix with time, status, Z columns

**References**

Wang, R., Lagakos, S.-W. and Gray, R.-J. Testing and interval estimation for two-sample survival comparisons with small sample sizes and unequal censoring. *Biostatistics*, 11(4), 676–692, January 2010.

---

sampleFromCondKM	<i>sampleFromCondKM</i>
------------------	-------------------------

---

**Description**

Sample from conditional distribution estimated by Kaplan-Meier estimator. Imputed values > tmax are right-censored.

**Usage**

```
sampleFromCondKM(U, fit, tmax = NULL, dv = 1, f = NULL)
```

**Arguments**

U	vector of observed times
fit	Kaplan-Meier fit as returned by survfit
tmax	largest observation of the pooled sample
dv	1 if imputing events, 0 if imputing censoring times
f	interpolated Kaplan-Meier estimate

**Value**

Random sample of survival times drawn from conditional distribution of T given  $T > U$

---

sampleFromKM	<i>sampleFromKM</i>
--------------	---------------------

---

**Description**

Sample from distribution estimated by Kaplan-Meier estimator. Imputed values > tmax are right-censored.

**Usage**

```
sampleFromKM(n, fit, start = 0, tmax = NULL, dv = 1)
```

**Arguments**

n	sample size
fit	Kaplan-Meier fit as returned by survfit
start	if 0 sample from L(T), else sample from L(T, T > start)
tmax	largest observation in pooled sample
dv	1 if imputing events, 0 if imputing censoring times

**Value**

Random sample of survival times

---

shuffleBlock	<i>shuffleBlock</i> Permute block preserving group sizes, randomization blocks
--------------	--

---

**Description**

shuffleBlock Permute block preserving group sizes, randomization blocks

**Usage**

```
shuffleBlock(block, strata = 0)
```

**Arguments**

block	vector of row indices to be permuted
strata	factor defining strata with block

**Value**

random permutation of each stratum within block

---

summary.permGS	<i>summary of permGS object</i>
----------------	---------------------------------

---

**Description**

summary of permGS object

**Usage**

```
## S3 method for class 'permGS'
summary(object, ...)
```

**Arguments**

- object            permGS object as returned by [createPermGS](#)
- ...              additional parameters (currently unused)

**Value**

nothing

# Index

coxph, [3](#), [6](#), [9–12](#)  
createPermGS, [2](#), [6](#), [16](#)  
  
exactLR, [3](#)  
  
imputeHeinze, [4](#)  
imputeIPT, [4](#)  
imputeIPZ, [5](#)  
  
nextStage, [6](#)  
  
parseFormula, [7](#)  
permGS, [7](#)  
permGS-package (permGS), [7](#)  
permHeinze, [8](#)  
permIPT, [9](#)  
permIPZ, [10](#)  
permLR, [11](#)  
permuteHeinze, [12](#)  
permuteIPT, [13](#)  
permuteIPZ, [13](#)  
  
sampleFromCondKM, [14](#)  
sampleFromKM, [15](#)  
shuffleBlock, [15](#)  
summary.permGS, [16](#)