Package 'fPASS'

July 22, 2025

Title Power and Sample Size for Projection Test under Repeated Measures

Version 1.0.0

Description Computes the power and sample size (PASS) required to test for the difference in the mean function between two groups under a repeatedly measured longitudinal or sparse functional design. See the manuscript by Koner and Luo (2023) <https://salilkoner.github.io/assets/PASS_manuscript.pdf> for details of the PASS formula and computational details. The details of the testing procedure for univariate and multivariate response are presented in Wang (2021) <doi:10.1214/21-EJS1802> and Koner and Luo (2023) <doi:10.48550/arXiv.2302.05612> respectively.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports dplyr, purrr, face, magrittr, MASS, Matrix, nlme, testthat, mgcv, lifecycle, expm, gamm4, gss, rlang, stringr, utils

Suggests knitr, rmarkdown, Hotelling, refund, foreach

VignetteBuilder knitr

URL https://github.com/SalilKoner/fPASS

BugReports https://github.com/SalilKoner/fPASS/issues

Config/testthat/edition 3

NeedsCompilation no

Author Salil Koner [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-1952-4210>), Sheng Luo [ctb, fnd]

Maintainer Salil Koner <salil.koner@duke.edu>

Repository CRAN

Date/Publication 2023-07-19 11:20:05 UTC

Contents

`Extract_Eigencomp_fDA` *Extract/estimate eigenfunction from a sparse functional or longitudinal design by simulating from a large number of subjects.*

Description

[Stable]

The function `Extract_Eigencomp_fDA()` computes the eigenfunctions and the covariance of the shrinkage scores required to conduct the projection-based test of mean function between two groups of longitudinal data or sparsely observed functional data under a random irregular design, as developed by Wang (2021).

Usage

```
Extract_Eigencomp_fDA(
  nobs_per_subj,
  obs.design,
  mean_diff_fnm,
  cov.type = c("ST", "NS"),
  cov.par,
  sigma2.e,
  missing_type = c("nomiss", "constant"),
  missing_percent = 0,
  eval_SS = 5000,
  alloc.ratio = c(1, 1),
  fpca_method = c("fpca.sc", "face"),
  work.grid = NULL,
  nWgrid = ifelse(is.null(work.grid), 101, length(work.grid)),
  data.driven.scores = FALSE,
  mean_diff_add_args = list(),
  fpca_optns = list()
)
```

Arguments

nobs_per_subj	The number of observations per subject. Each element of it must be greater than 3. It could also be a vector to indicate that the number of observation for each is randomly varying between the elements of the vector, or a scalar to ensure that the number of observations are same for each subject. See examples.
obs.design	The sampling design of the observations. Must be provided as a list with the following elements. If the design is longitudinal (e.g. a clinical trial where there is pre-specified schedule of visit for the participants) it must be a named list with elements design, visit.schedule and visit.window, where obs.design$design must be specified as 'longitudinal', visit.schedule specifying schedule of visits (in months or days or any unit of time), other than the baseline visit and visit.window denoting the maximum time window for every visit. For functional design (where the observation points are either densely observed within a compact interval or under a sparse random design), the argument must be provided as a named list with elements design and fun.domain, where obs.design$design must be specified as 'functional' and obs.design$fun.domain must be specified as a two length vector indicating the domain of the function. See Details on the specification of arguments section below more details.
mean_diff_fnm	The name of the function that output of the difference of the mean between the two groups at any given time. It must be supplied as character, so that match.fun(mean_diff_fnm) returns a valid function, that takes a vector input, and returns a vector of the same length of the input.
cov.type	The type of the covariance structure of the data, must be either of 'ST' (stationary) or 'NS' (non-stationary). This argument along with the cov.par argument must be specified compatibly to ensure that the function does not return an error. See the details of cov.par argument.
cov.par	The covariance structure of the latent response trajectory. If cov.type == 'ST' then, cov.par must be specified a named list of two elements, var and cor, where var is the common variance of the observations, which must be a positive number; and cor specifies the correlation structure between the observations. cov.par$cor must be specified in the form of the nlme::corClasses specified in R package **nlme**. Check the package documentation for more details for each of the correlation classes. The cov.par$cor must be a corStruct class so it can be passed onto the nlme::corMatrix() to extract the subject-specific covariance matrix. If cov.type='NS' then, cov.par must be a named list of two elements, cov.obj and eigen.comp, where only one of the cov.par$cov.obj or cov.par$eigen.comp must be non-null. This is to specify that the covariance structure of the latent trajectory can be either provided in the form of covariance function or in the form of eigenfunction and eigenvalues (Spectral decomposition). If the cov.par$cov.obj is specified, then it must be a bivariate function, with two arguments. Alternatively, if the true eigenfunctions are known, then the user can specify that by specifying cov.par$eigen.comp. In this case, the cov.par$eigen.comp must be a named list with two elements, eig.obj and eig.val, where cov.par$eigen.comp$eig.val must be positive vector and cov.par$eigen.comp$eig.obj must be a vectorized function so that its evaluation at a vector of time points returns a matrix of dimension r by length(cov.par$eigen.comp$eig.val), with r being the length of time

points.

`sigma2.e` Measurement error variance, should be set as zero or a very small number if the measurement error is not significant.

`missing_type` The type of missing in the number of observations of the subjects. Can be one of `'nomiss'` for no missing observations or `'constant'` for constant missing percentage at every time point. The current version of package only supports `missing_type = 'constant'`.

`missing_percent`
The percentage of missing at each observation points for each subject. Must be supplied as number between [0, 0.8], as missing percentage more than 80% is not practical. If `nobs_per_subj` is supplied as vector, then `missing_type` is forced to set as `'nomiss'` and `missing_percent = 0`, because the `missing_type = 'constant'` has no meaning if the number of observations are varying between the subject at the first, typically considered in the case of sparse random functional design.

`eval_SS` The sample size based on which the eigencomponents will be estimated from data. To compute the theoretical power of the test we must make sure that we use a large enough sample size to generate the data such that the estimated eigenfunctions are very close to the true eigenfunctions and that the sampling design will not have much effect on the loss of precision. Default value 5000.

`alloc.ratio` The allocation ratio of samples in the each group. Note that the eigenfunctions will still be estimated based on the total sample_size, however, the variance of the `shrinkage` scores (which is required to compute the power function) will be estimated based on the allocation of the samples in each group. Must be given as vector of length 2. Default value is set at `c(1, 1)`, indicating equal sample size.

`fpca_method` The method by which the FPCA is computed. Must be one of 'fpca.sc' and 'face'. If `fpca_method == 'fpca.sc'` then the eigencomponents are estimated using the function `refund::fpca.sc()`. However, since the `refund::fpca.sc()` function fails to estimate the correct `shrinkage` scores, and throws `NA` values when the measurement errors is estimated to be zero, we wrote out a similar function where we corrected those error in current version of `refund::fpca.sc()`. Check out the `fpca_sc()` function for details. If `fpca_method == 'face'`, then the eigencomponents are estimated using `face::face.sparse()` function.

`work.grid` The working grid in the domain of the functions, where the eigenfunctions and other covariance components will be estimated. Default is NULL, then, a equidistant grid points of length `nWgrid` will be internally created to as the default `work.grid`.

`nWgrid` The length of the `work.grid` in the domain of the function based on which the eigenfunctions will be estimated. Default value is 101. If `work.grid` is specified, then `nWgrid` must be null, and vice-versa.

`data.driven.scores`
Indicates whether the scores are estimated from the full data, WITHOUT assuming the mean function is unknown, rather the mean function is estimated using `mgcv::gam()` function.

`mean_diff_add_args`
: Additional arguments to be passed to group difference function specified in the argument `mean_diff_fnm`.

`fpca_optns`
: Additional options to be passed onto either of `fpca_sc()` or `face::face.sparse()` function in order to estimate the eigencomponents. It must be a named list with elements to be passed onto the respective function, depending on the `fpca_method`. The names of the list must not match either of `c('data', 'newdata', 'argvals.new')` for `fpca_method == 'face'` and must not match either of `c('ydata', 'Y.pred')` for `fpca_method == 'fpca.sc'`.

Details

The function can handle data from wide variety of covariance structure, can be parametric, or non-parametric. Additional with traditional stationary structures assumed for longitudinal data (see nlme::corClasses), the user can specify any other non-stationary covariance function in the form of either a covariance function or in terms of eigenfunctions and eigenvalues. The user have a lot of flexibility into tweaking the arguments `nobs_per_subject`, `obs.design`, and `cov.par` to compute the eigencomponents under different sampling design and covariance process of the response trajectory, and for any arbitrary mean difference function. Internally, using the sampling design and the covariance structure specified, we generate a large data with large number of subjects, and estimate the eigenfunctions and the covariance of the estimated `shrinkage` scores by means of functional principal component analysis (fPCA). We put the option of using two most commonly used softwares for fPCA in the functional data literature, `refund::fpca.sc()` and `face::face.sparse()`. However, since the `refund::fpca.sc()` do not compute the `shrinkage` scores correctly, especially when the measurement error variance is estimated to be zero, we made a duplicate version of that function in our package, where we write out the scoring part on our own. The new function is named as `fpca_sc()`, please check it out.

Value

A list with the elements listed below.

1. `mean_diff_vec` - The evaluation of the mean function at the working grid.
2. `est_eigenfun` - The evaluation of the estimated eigenfunctions at the working grid.
3. `est_eigenval` - Estimated eigen values.
4. `working.grid` - The grid points at which `mean_diff_vec` and `est_eigenfun` are evaluated.
5. `fpcCall` - The exact call of either of the `fpca_sc()` or `face::face.sparse()` used to compute the eigencomponents.
6. `scores_var1` - Estimated covariance of the `shrinkage` scores for the treatment group.
7. `scores_var2` - Estimated covariance of the `shrinkage` scores for the placebo group.
8. `pooled_var` - Pooled covariance of the scores combining both the groups. This is required if the user wants to compute the power of Hotelling T statistic under equal variance assumption.

If `data.driven.scores == TRUE` additional components are returned

1. `scores_1` - Estimated `shrinkage` scores for all the subjects in treatment group.
2. `scores_2` - Estimated `shrinkage` scores for all the subjects in placebo group.

The output of this function is designed such a way the user can directly input the output obtained from this function into the arguments of `Power_Proj_Test_ufDA()` function to obtain the power and the sample size right away. The function PASS_Proj_Test_ufDA does the same, it is essentially a wrapper of`Extract_Eigencomp_fDA()` and `Power_Proj_Test_ufDA()` together.

Specification of key arguments

If `obs.design$design == 'functional'` then a dense grid of length, specified by ngrid (typically 101/201) is internally created, and the observation points will be randomly chosen from them. The time points could also randomly chosen between any number between the interval, but then for large number of subject, `fpca_sc()` function will take huge time to estimate the eigenfunction. For dense design, the user must set a large value of the argument `nobs_per_subj` and for sparse (random) design, `nobs_per_subj` should be set small (and varying). On the other hand, typical to longitudinal data, if the measurements are taken at fixed time points (from baseline) for each subject, then the user must set `obs.design$design == 'longitudinal'` and the time points must be accordingly specified in the argument `obs.design$visit.schedule`. The length of `obs.design$visit.schedule` must match `length(nobs_per_subj)-1`. Internally, when `obs.design$design == 'longitudinal'`, the function scale the visit times so that it lies between [0, 1], so the user should not specify any element named `fun.domain` in the list for `obs.design$design == 'longitudinal'`. Make sure that the mean function and the covariance function specified in the `cov.par` and `mean_diff_fnm` parameter also scaled to take argument between [0, 1]. Also, it is imperative to say that `nobs_per_subj` must be of a scalar positive integer for `design == 'longitudinal'`.

Author(s)

Salil Koner
Maintainer: Salil Koner <salil.koner@duke.edu>

References

Wang, Qiyao (2021) *Two-sample inference for sparse functional data, Electronic Journal of Statistics, Vol. 15, 1395-1423*
doi:10.1214/21EJS1802.

See Also

See `Power_Proj_Test_ufDA()`, `refund::fpca.sc()` and `face::face.sparse()`.

Examples

```
# Example 1: Extract eigencomponents from stationary covariance.

set.seed(12345)
mean.diff <- function(t) {t};
obs.design <- list("design" = "longitudinal",
"visit.schedule" = seq(0.1, 0.9, length.out=7),
"visit.window" = 0.05)
cor.str <- nlme::corExp(1, form = ~ time | Subject);
sigma2 <- 1; sigma2.e <- 0.25; nobs_per_subj <- 8;
missing_type <- "constant"; missing_percent <- 0.01;
```

```
eigencomp  <- Extract_Eigencomp_fDA(obs.design = obs.design,
mean_diff_fnm = "mean.diff", cov.type = "ST",
cov.par = list("var" = sigma2, "cor" = cor.str),
sigma2.e = sigma2.e, nobs_per_subj = nobs_per_subj,
missing_type = missing_type,
missing_percent = missing_percent, eval_SS = 1000,
alloc.ratio = c(1,1), nWgrid = 201,
fpca_method = "fpca.sc", data.driven.scores = FALSE,
mean_diff_add_args = list(), fpca_optns = list(pve = 0.95))

# Example 2: Extract eigencomponents from non-stationary covariance.

alloc.ratio  <- c(1,1)
mean.diff    <- function(t) {1 * (t^3)};
eig.fun <- function(t, k) { if (k==1) {
ef <- sqrt(2)*sin(2*pi*t)
} else if (k==2) {ef <- sqrt(2)*cos(2*pi*t)}
return(ef)}
eig.fun.vec  <- function(t){cbind(eig.fun(t, 1),eig.fun(t, 2))}
eigen.comp   <- list("eig.val" = c(1, 0.5), "eig.obj" = eig.fun.vec)
obs.design   <- list(design = "functional", fun.domain = c(0,1))
cov.par      <- list("cov.obj" = NULL, "eigen.comp" = eigen.comp)
sigma2.e     <- 0.001; nobs_per_subj <- 4:7;
missing_type <- "nomiss"; missing_percent <- 0;
fpca_method  <- "fpca.sc"
eigencomp  <- Extract_Eigencomp_fDA(obs.design = obs.design,
 mean_diff_fnm = "mean.diff",
 cov.type = "NS", cov.par = cov.par,
 sigma2.e = sigma2.e, nobs_per_subj = nobs_per_subj,
 missing_type = missing_type,
 missing_percent = missing_percent, eval_SS = 1000,
 alloc.ratio = alloc.ratio, nWgrid = 201,
 fpca_method = "fpca.sc", data.driven.scores = FALSE,
 mean_diff_add_args = list(), fpca_optns = list(pve = 0.95))
```

PASS_Proj_Test_ufDA *Power and Sample size (PASS) calculation of Two-Sample Projection-based test for sparsely observed univariate functional data.*

Description

[Stable]

The function PASS_Proj_Test_ufDA() computes the power and sample size (PASS) required to conduct the projection-based test of mean function between two groups of longitudinal data or sparsely observed functional data under a random irregular design, under common covariance structure between the groups. See Wang (2021) for more details of the testing procedure.

Usage

```
PASS_Proj_Test_ufDA(
  sample_size,
  target.power,
  sig.level = 0.05,
  nobs_per_subj,
  obs.design,
  mean_diff_fnm,
  cov.type = c("ST", "NS"),
  cov.par,
  sigma2.e,
  missing_type = c("nomiss", "constant"),
  missing_percent = 0,
  eval_SS = 5000,
  alloc.ratio = c(1, 1),
  fpca_method = c("fpca.sc", "face"),
  mean_diff_add_args = list(),
  fpca_optns = list(pve = 0.95),
  nWgrid = 201,
  npc_to_use = NULL,
  return.eigencomp = FALSE,
  nsim = 10000
)
```

Arguments

`sample_size`	Total sample size combining both the groups, must be a positive integer.
`target.power`	Target power to achieve, must be a number between 0 and 1. Only one of `sample_size` and `target.power` should be non-null. The function will return sample size if `sample_size` is NULL, and return power if `target.power` is NULL.
`sig.level`	Significance level of the test, default set at 0.05, must be less than 0.2.
`nobs_per_subj`	The number of observations per subject. Each element of it must be greater than 3. It could also be a vector to indicate that the number of observation for each is randomly varying between the elements of the vector, or a scalar to ensure that the number of observations are same for each subject. See examples.
`obs.design`	The sampling design of the observations. Must be provided as a list with the following elements. If the design is longitudinal (e.g. a clinical trial where there is pre-specified schedule of visit for the participants) it must be a named list with elements `design`, `visit.schedule` and `visit.window`, where `obs.design$design` must be specified as `'longitudinal'`, `visit.schedule` specifying schedule of visits (in months or days or any unit of time), other than the baseline visit and `visit.window` denoting the maximum time window for every visit. For functional design (where the observation points are either densely observed within a compact interval or under a sparse random design), the argument must be provided as a named list with elements `design` and `fun.domain`, where `obs.design$design` must be specified as `'functional'` and `obs.design$fun.domain`

must be specified as a two length vector indicating the domain of the function. See Details on the specification of arguments section below more details.

`mean_diff_fnm` The name of the function that output of the difference of the mean between the two groups at any given time. It must be supplied as character, so that `match.fun(mean_diff_fnm)` returns a valid function, that takes a vector input, and returns a vector of the same length of the input.

`cov.type` The type of the covariance structure of the data, must be either of 'ST' (stationary) or 'NS' (non-stationary). This argument along with the `cov.par` argument must be specified compatibly to ensure that the function does not return an error. See the details of `cov.par` argument.

`cov.par` The covariance structure of the latent response trajectory. If `cov.type == 'ST'` then, `cov.par` must be specified a named list of two elements, `var` and `cor`, where `var` is the common variance of the observations, which must be a positive number; and `cor` specifies the correlation structure between the observations. `cov.par$cor` must be specified in the form of the nlme::corClasses specified in R package **nlme**. Check the package documentation for more details for each of the correlation classes. The `cov.par$cor` must be a `corStruct` class so it can be passed onto the `nlme::corMatrix()` to extract the subject-specific covariance matrix. If `cov.type='NS'` then, `cov.par` must be a named list of two elements, `cov.obj` and `eigen.comp`, where only one of the `cov.par$cov.obj` or `cov.par$eigen.comp` must be non-null. This is to specify that the covariance structure of the latent trajectory can be either provided in the form of covariance function or in the form of eigenfunction and eigenvalues (Spectral decomposition). If the `cov.par$cov.obj` is specified, then it must be a bivariate function, with two arguments. Alternatively, if the true eigenfunctions are known, then the user can specify that by specifying `cov.par$eigen.comp`. In this case, the `cov.par$eigen.comp` must be a named list with two elements, `eig.obj` and `eig.val`, where `cov.par$eigen.comp$eig.val` must be positive vector and `cov.par$eigen.comp$eig.obj` must be a vectorized function so that its evaluation at a vector of time points returns a matrix of dimension r by `length(cov.par$eigen.comp$eig.val)`, with r being the length of time points.

`sigma2.e` Measurement error variance, should be set as zero or a very small number if the measurement error is not significant.

`missing_type` The type of missing in the number of observations of the subjects. Can be one of `'nomiss'` for no missing observations or `'constant'` for constant missing percentage at every time point. The current version of package only supports `missing_type = 'constant'`.

`missing_percent`

The percentage of missing at each observation points for each subject. Must be supplied as number between [0, 0.8], as missing percentage more than 80% is not practical. If `nobs_per_subj` is supplied as vector, then `missing_type` is forced to set as `'nomiss'` and `missing_percent = 0`, because the `missing_type = 'constant'` has no meaning if the number of observations are varying between the subject at the first, typically considered in the case of sparse random functional design.

`eval_SS` The sample size based on which the eigencomponents will be estimated from data. To compute the theoretical power of the test we must make sure that we use a large enough sample size to generate the data such that the estimated eigenfunctions are very close to the true eigenfunctions and that the sampling design will not have much effect on the loss of precision. Default value 5000.

`alloc.ratio` The allocation ratio of samples in the each group. Note that the eigenfunctions will still be estimated based on the total sample_size, however, the variance of the `shrinkage` scores (which is required to compute the power function) will be estimated based on the allocation of the samples in each group. Must be given as vector of length 2. Default value is set at `c(1, 1)`, indicating equal sample size.

`fpca_method` The method by which the FPCA is computed. Must be one of 'fpca.sc' and 'face'. If `fpca_method == 'fpca.sc'` then the eigencomponents are estimated using the function `refund::fpca.sc()`. However, since the `refund::fpca.sc()` function fails to estimate the correct `shrinkage` scores, and throws `NA` values when the measurement errors is estimated to be zero, we wrote out a similar function where we corrected those error in current version of `refund::fpca.sc()`. Check out the `fpca_sc()` function for details. If `fpca_method == 'face'`, then the eigencomponents are estimated using `face::face.sparse()` function.

`mean_diff_add_args` Additional arguments to be passed to group difference function specified in the argument `mean_diff_fnm`.

`fpca_optns` Additional options to be passed onto either of `fpca_sc()` or `face::face.sparse()` function in order to estimate the eigencomponents. It must be a named list with elements to be passed onto the respective function, depending on the `fpca_method`. The names of the list must not match either of `c('data', 'newdata', 'argvals.new')` for `fpca_method == 'face'` and must not match either of `c('ydata', 'Y.pred')` for `fpca_method == 'fpca.sc'`.

`nWgrid` The length of the working grid based in the domain of the function on which the eigenfunctions will be estimated. The actual working grid will be calculated using the `gss::gauss.quad()` function (so that it facilitates the numerical integration of the eigenfunction with the mean function using gaussian quadrature rule)

`npc_to_use` Number of eigenfunctions to use to compute the power. Default is NULL, in which case all the eigenfunctions estimated from the data will be used.

`return.eigencomp` Indicates whether to return the eigencomponents obtained from the fPCA on the large data with sample size equal to `eval_SS`. Default is `FALSE`.

`nsim` The number of samples to be generated from the alternate distribution of Hotelling T statistic. Default value is 10000.

Details

The function is designed to perform the power and sample size analysis for functional under a dense and sparse (random) design and longitudinal data. The function can handle data from wide variety of covariance structure, can be parametric, or non-parametric. Additional with traditional stationary structures assumed for longitudinal data (see nlme::corClasses), the user can specify any

other non-stationary covariance function in the form of either a covariance function or in terms of eigenfunctions and eigenvalues. The user have a lot of flexibility into tweaking the arguments of the function to assess the power function of the test under different sampling design and covariance process of the response trajectory, and for any arbitrary mean difference function. Overall, the functionality of the module is quite comprehensive and includes all the different cases considered in the 'NCSS PASS (2023)' software. We believe that this software can be an effective clinical trial design tools when considering the projection-based test as the primary decision making method.

Value

A list with following elements, `power_value` if `is.null(target.power)` then returns the power of the test when n equal to `sample_size`, otherwise `required_SS`, the sample size required to achieve the power of the test at `target.power`. If `return.eigencomp == TRUE` then `est_eigencomp` is also returned, containing the entire output obtained from internal call of `Extract_Eigencomp_fDA()`.

Specification of key arguments

If `obs.design$design == 'functional'` then a dense grid of length, specified by ngrid (typically 101/201) is internally created, and the observation points will be randomly chosen from them. The time points could also randomly chosen between any number between the interval, but then for large number of subject, `fpca_sc()` function will take huge time to estimate the eigenfunction. For dense design, the user must set a large value of the argument `nobs_per_subj` and for sparse (random) design, `nobs_per_subj` should be set small (and varying). On the other hand, typical to longitudinal data, if the measurements are taken at fixed time points (from baseline) for each subject, then the user must set `obs.design$design == 'longitudinal'` and the time points must be accordingly specified in the argument `obs.design$visit.schedule`. The length of `obs.design$visit.schedule` must match `length(nobs_per_subj)-1`. Internally, when `obs.design$design == 'longitudinal'`, the function scale the visit times so that it lies between [0, 1], so the user should not specify any element named `fun.domain` in the list for `obs.design$design == 'longitudinal'`. Make sure that the mean function and the covariance function specified in the `cov.par` and `mean_diff_fnm` parameter also scaled to take argument between [0, 1]. Also, it is imperative to say that `nobs_per_subj` must be of a scalar positive integer for `design == 'longitudinal'`.

Author(s)

Salil Koner
Maintainer: Salil Koner <salil.koner@duke.edu>

References

Wang, Qiyao (2021) *Two-sample inference for sparse functional data, Electronic Journal of Statistics, Vol. 15, 1395-1423* doi:10.1214/21EJS1802.

PASS 2023 Power Analysis and Sample Size Software (2023). NCSS, LLC. Kaysville, Utah, USA, ncss.com/software/pass.

See Also

See `Power_Proj_Test_ufDA()` and `Extract_Eigencomp_fDA()`.

Examples

```
# Example 1: Power analysis for stationary exponential covariance.
# Should return a power same as the size because
# the true mean difference is zero.

set.seed(12345)
mean.diff <- function(t) {0*t};
obs.design = list("design" = "longitudinal",
                  "visit.schedule" = seq(0.1, 0.9, length.out=7),
                   "visit.window" = 0.05)
cor.str <- nlme::corExp(1, form = ~ time | Subject);
sigma2 <- 1; sigma2.e <- 0.25; nobs_per_subj <- 8;
missing_type <- "constant"; missing_percent <- 0.01;
# Please increase `eval_SS` argument from 1000 to 5000 to get
# accurate precision on the estimated eigenfunctions.
pow  <- PASS_Proj_Test_ufDA(sample_size = 100, target.power = NULL, sig.level = 0.05,
                            obs.design = obs.design,
                            mean_diff_fnm = "mean.diff", cov.type = "ST",
                            cov.par = list("var" = sigma2, "cor" = cor.str),
                            sigma2.e = sigma2.e, nobs_per_subj = nobs_per_subj,
                            missing_type = missing_type,
                            missing_percent = missing_percent, eval_SS = 1000,
                            alloc.ratio = c(1,1), nWgrid = 201,
                            fpca_method = "fpca.sc",
                           mean_diff_add_args = list(), fpca_optns = list("pve" = 0.95),
                            nsim = 1e3)

print(pow$power_value)

# Example 2: Sample size calculation for a non-stationary covariance:

alloc.ratio  <- c(1,1)
mean.diff    <- function(t) {3 * (t^3)};
eig.fun <- function(t, k) {
  if (k==1) ef <- sqrt(2)*sin(2*pi*t)
  else if (k==2) ef <- sqrt(2)*cos(2*pi*t)
  return(ef)}
eig.fun.vec  <- function(t){cbind(eig.fun(t, 1),eig.fun(t, 2))}
eigen.comp   <- list("eig.val" = c(1, 0.5), "eig.obj" = eig.fun.vec)
obs.design   <- list(design = "functional", fun.domain = c(0,1))
cov.par      <- list("cov.obj" = NULL, "eigen.comp" = eigen.comp)
sigma2.e     <- 0.001; nobs_per_subj <- 4:7;
missing_type <- "nomiss"; missing_percent <- 0;
fpca_method  <- "fpca.sc"
# Please increase `eval_SS` argument from 1000 to 5000 to get
# accurate precision on the estimated eigenfunctions.
pow  <- PASS_Proj_Test_ufDA(sample_size = NULL, target.power = 0.8,
                            sig.level = 0.05, obs.design = obs.design,
                            mean_diff_fnm = "mean.diff", cov.type = "NS",
                            cov.par = cov.par, sigma2.e = sigma2.e,
                            nobs_per_subj = nobs_per_subj, missing_type = missing_type,
                            missing_percent = missing_percent, eval_SS = 1000,
```

```
                          alloc.ratio = alloc.ratio, fpca_method = "fpca.sc",
                          mean_diff_add_args = list(), fpca_optns = list(pve = 0.95),
                          nsim = 1e3, nWgrid = 201)

print(pow$required_SS)
```

---

pHotellingT *CDF of Hotelling-T^2 statistic.*

Description

[Stable]

The function `pHotellingT()` computes the cumulative distribution function (CDF) of the two-sample Hotelling-T^2 statistic ($P(T > q)$) in the multivariate response setting. This function is used to compute the power function of Two-Sample (TS) Projection-based test (Wang 2021, EJS.) for sparsely observed univariate functional data.

Usage

```
pHotellingT(
  q,
  total_sample_size,
  mean_diff,
  sig1,
  sig2,
  alloc.ratio = c(1, 1),
  lower.tail = TRUE,
  nsim = 10000
)
```

Arguments

`q`	The point at which the CDF needs to be evaluated
`total_sample_size`	Target sample size, must be a positive integer.
`mean_diff`	The difference in the mean vector between the two groups, must be a vector.
`sig1`	The true (or estimate) of covariance matrix for the first group. Must be symmetric (`is.symmetric(sig1) == TRUE`) and positive definite (`chol(sig1)` without an error!).
`sig2`	The true (or estimate) of covariance matrix for the second group. Must be symmetric (`is.symmetric(sig2) == TRUE`) and positive definite (`chol(sig2)` without an error!).
`alloc.ratio`	Allocation of total sample size into the two groups. Must set as a vector of two positive numbers. For equal allocation it should be put as c(1,1), for non-equal allocation one can put c(2,1) or c(3,1) etc.
`lower.tail`	if TRUE, the CDF is returned, otherwise right tail probability is returned.
`nsim`	The number of samples to be generated from the alternate distribution.

Details

Under the assumption of the equal variance, we know that the alternative distribution of the Hotelling-T^2 statistic $((n-k-1)T/(n-2)*K)$ has an F distribution with the non-centrality depending on the difference between the true mean vectors and the (common) covariance of the response. However, when the true covariance of the true groups of responses differ, the alternate distribution becomes non-trivial. Koner and Luo (2023) proved that the alternate distribution of the test-statistic approximately follows a ratio of the linear combination of the K (dimension of the response) non-central chi-squared random variables (where the non-centrality parameter depends on the mean difference) and a chi-squared distribution whose degrees of freedom depends on a complicated functions of sample size in the two groups. This function initially calls the Sim_HotellingT_unequal_var function to obtain the samples from the non-null distribution and computes the CDF numerically with high precision based on a large number of samples. See Koner and Luo (2023) for more details on the formula of the non-null distribution.

Value

The CDF of the Hotelling T statistic, if `lower.tail == TRUE`, otherwise the right tail probability is returned.

Author(s)

Salil Koner
Maintainer: Salil Koner <salil.koner@duke.edu>

See Also

`Hotelling::hotelling.test()`, `Hotelling::hotelling.stat()` to generate empirical samples from the Hotelling T-statistic from empirical data.

Examples

```
B           <- 10000
k           <- 4
n2          <- 60
n1_by_n2    <- 2
n1          <- n1_by_n2 * n2
mu1         <- rep(0,k)
del         <- 0.4
mu2         <- mu1 + rep(del, k) # rep(0.19,k)  # 0.23 (0.9), 0.18 (0.7) 0.20 (0.8)
sig1        <- diag(k)
sig2        <- sig1
cutoff      <- seq(0,30, length.out=20)
the_cdf     <- round(pHotellingT(cutoff, n1+n2, mu1 - mu2,
                                 sig1, sig2, alloc.ratio=c(2,1),
                                 lower.tail=FALSE, nsim = 1e4),3)
```

Power_Proj_Test_ufDA *Power of the Two-sample Projection-based test for functional data with known (or estimated) eigencomponents.*

Description

[Stable]

The function Power_Proj_Test_ufDA() computes the power of of the two-sample projection-based test for functional response data setting, when the group difference, the eigenfunctions of the covariance of the data are specified at dense grid of time points, along with the (estimated) covariance of the shrinkage scores.

Usage

```
Power_Proj_Test_ufDA(
  total_sample_size,
  argvals,
  mean_vector,
  eigen_matrix,
  scores_var1,
  scores_var2,
  weights,
  sig.level = 0.05,
  alloc.ratio = c(1, 1),
  npc_to_pick = ncol(eigen_matrix),
  nsim = 10000
)
```

Arguments

total_sample_size	Total sample size combing the two groups, must be a positive integer.
argvals	The working grid of timepoints to evaluate the eigenfunctions and the mean functions. It is preferred to take the working grid as dense grid so that $\int[\mu_1(t) - \mu_2(t)]\phi_k(t)\,dt$ can be calculated with a required precision.
mean_vector	The difference in the mean function evaluated at argvals, must be a numeric vector of length same as that that of argavls.
eigen_matrix	The matrix of eigenfunctions evaluated at argvals, must be a length(argvals) by K matrix, where K is the number of eigenfunctions.
scores_var1	The true (or estimate) of covariance matrix of the shrinkage scores for the first group. Must be symmetric (is.symmetric(scores_var1) == TRUE) and positive definite (chol(scores_var1) without an error!).
scores_var2	The true (or estimate) of covariance matrix of the shrinkage scores for the second group. Must be symmetric (is.symmetric(scores_var2) == TRUE) and positive definite (chol(scores_var2) without an error!).

`weights` The weights to put to compute the projection $\int [\mu_1(t) - \mu_2(t)]\phi_k(t)\, dt$, for each $k = 1, \ldots, K$. The integral is numerically approximated as `sum(mean_diff(argvals)*eigen_matrix[,`

`sig.level` Significance level of the test, default set at 0.05, must be less than 0.2.

`alloc.ratio` The allocation ratio of samples in the each group. Note that the eigenfunctions will still be estimated based on the total sample_size, however, the variance of the `shrinkage` scores (which is required to compute the power function) will be estimated based on the allocation of the samples in each group. Must be given as vector of length 2. Default value is set at `c(1, 1)`, indicating equal sample size.

`npc_to_pick` Number of eigenfunction to be used to compute the power. Typically this is becomes handy when the user want to discard few of the last eigenfunctions, typically with a very small eigenvalues.

`nsim` The number of samples to be generated from the alternate distribution of Hotelling T statistic. Default value is 10000.

Details

The projection-based test first extracts K eigenfunctions from the data, and then project the mean difference function onto each of the eigenfunctions to obtain a K-dimensional projection vector that reflects the group difference. Wang (2021) pointed that under the null hypothesis the covariance of K-dimensional functional principal component analysis (fPCA) scores are the same, and thus a Hotelling T^2 test with assuming equal variance of the shrinkage scores is a valid test. However, Koner and Luo (2023) pointed out that under the alternate hypothesis, when the difference is mean is significant, the covariance of the shrinkage scores also differ between the groups. Therefore, while computing the power of test, we must have to derive the distribution of the Hotelling T^2 statistic under the assumption of unequal variance. The alogrithm for the power of multivariate Hotelling T^2 under unequal variance is coded in `pHotellingT()` function. This particular function is a wrapper around that function, which inputs the mean difference as a function, and the eigenfunctions and the scores, and subsequently call the `pHotellingT()` function to compute the power under unequal variance. See Koner and Luo (2023) for more details on the formula of the non-null distribution.

Value

Power of the projection-based test for specified difference in the mean function and the eigencomponents of the covariance of the functional data.

Author(s)

Salil Koner
Maintainer: Salil Koner <salil.koner@duke.edu>

References

Wang, Qiyao (2021) *Two-sample inference for sparse functional data, Electronic Journal of Statistics, Vol. 15, 1395-1423*
doi:10.1214/21EJS1802.

See Also

See pHotellingT() and Sim_HotellingT_unequal_var() for samples from Hotelling T distribution.

Examples

```
ngrid          <- 101
interval       <- c(-1,1)
gauss.quad.pts <- gss::gauss.quad(ngrid,interval) # evaluation points
working.grid   <- gauss.quad.pts$pt
mean_fn        <- function(t) {0.4*sin(2*pi*t)}
mean_vector    <- mean_fn(working.grid)
eigen_fn       <- function(t, k){ sqrt(2)*{(k==2)*sin(2*pi*t) + (k==1)*cos(2*pi*t)} }
eigen_matrix   <- cbind(eigen_fn(working.grid,1), eigen_fn(working.grid,2))
mean_proj      <- sapply(1:2, function(r) integrate(function(x)
eigen_fn(x,r)*mean_fn(x), interval[1], interval[2])$value)
sig1           <- diag(2)
sig2           <- 2*diag(2)
alp            <- 0.05
n              <- 100
k              <- ncol(eigen_matrix)
cutoff         <- {(n - 2)*k/(n - k -1)}*qf(1-alp, k, n-k-1)
func_power     <- Power_Proj_Test_ufDA(total_sample_size=n,
argvals=working.grid,
mean_vector = mean_vector, eigen_matrix = eigen_matrix,
scores_var1 = sig1, scores_var2= sig2, weights = gauss.quad.pts$wt,
sig.level=alp, alloc.ratio = c(1,1), npc_to_pick=ncol(eigen_matrix),
nsim = 5e3)
```

`Sim_HotellingT_unequal_var` *Samples from the non-null distribution of the Hotelling-T^2 statistic under unequal covariance.*

Description

[Stable]

The function `Sim_HotellingT_unequal_var()` generates samples from the (non-null) distribution of the two-sample Hotelling-T^2 statistic under the assuming of unequal covariance of the multivariate response between the two groups. This function is used to compute the power function of Two-Sample (TS) Projection-based test (Wang 2021, EJS.) for sparsely observed univariate functional data.

Usage

```
Sim_HotellingT_unequal_var(
  total_sample_size,
```

```
  mean_diff,
  sig1,
  sig2,
  alloc.ratio = c(1, 1),
  nsim = 10000
)
```

Arguments

`total_sample_size`	Target sample size, must be a positive integer.
`mean_diff`	The difference in the mean vector between the two groups, must be a vector.
`sig1`	The true (or estimate) of covariance matrix for the first group. Must be symmetric (`is.symmetric(sig1) == TRUE`) and positive definite (`chol(sig1)` without an error!).
`sig2`	The true (or estimate) of covariance matrix for the second group. Must be symmetric (`is.symmetric(sig2) == TRUE`) and positive definite (`chol(sig2)` without an error!).
`alloc.ratio`	Allocation of total sample size into the two groups. Must set as a vector of two positive numbers. For equal allocation it should be put as c(1,1), for non-equal allocation one can put c(2,1) or c(3,1) etc.
`nsim`	The number of samples to be generated from the alternate distribution.

Details

Under the assumption of the equal variance, we know that the alternative distribution of the Hotelling-T^2 statistic has an F distribution with the non-centrality depending on the difference between the true mean vectors and the (common) covariance of the response. However, when the true covariance of the true groups of responses differ, the alternate distribution becomes non-trivial. Koner and Luo (2023) proved that the alternate distribution of the test-statistic approximately follows a ratio of the linear combination of the K (dimension of the response) non-central chi-squared random variables (where the non-centrality parameter depends on the mean difference) and a chi-squared distribution whose degrees of freedom depends on a complicated functions of sample size in the two groups. See Koner and Luo (2023) for more details on the formula of the non-null distribution.

Value

A named list with two elements.

1. `samples` - a vector of length `nsim`, containing The samples from the distribution of the Hotelling T statistic under unequal variance.
2. `denom.df` - The denominator degrees of freedom of the chi-square statistic obtained by approximation of the sum of two Wishart distribution under unequal variance.

Author(s)

Salil Koner
Maintainer: Salil Koner <salil.koner@duke.edu>

References

Wang, Qiyao (2021) *Two-sample inference for sparse functional data, Electronic Journal of Statistics, Vol. 15, 1395-1423*
doi:10.1214/21EJS1802.

See Also

`Hotelling::hotelling.test()`, `Hotelling::hotelling.stat()` to generate empirical samples from the Hotelling T-statistic from empirical data.

Examples

```
# Case 1: Null hypothesis is true. True mean difference is zero, and the true
# covariance of the two groups are same.
k <- 5
mu1  <- rep(0,k); del  <- 0; mu2 <- mu1 + rep(del, k);
sig1 <- diag(k); sig2 <- sig1 + del*toeplitz(c(1,rep(0.5, k-1))); n <- 200;
null.dist.samples <- Sim_HotellingT_unequal_var(total_sample_size=n, mean_diff=mu1-mu2,
                     sig1=sig1, sig2=sig2, alloc.ratio=c(1,1), nsim=1e3)
# The following Kolmogorov Smirnov test confirms that under null hypothesis
# and when the covariances are same, the distribution is exactly a
# central F distribution with \eqn{k} and \eqn{n-k}  degrees of freedom.
ks.test(null.dist.samples$samples, {{(n - 2) * k}/(n - k -1)} * {rf(n=1e3, k, n-k-1)} )


# Case 2: Alternate hypothesis is true. The mean difference is non-zero,
# and the covariances of the two groups are same:
k <- 6
mu1  <- rep(0,k); del  <- 0.15; mu2 <- mu1 + rep(del, k);
sig1 <- diag(k); sig2 <- sig1;
n1 <- 100; n2 <- 100;
alt.dist.samples <- Sim_HotellingT_unequal_var(total_sample_size=n1+n2, mean_diff=mu1-mu2,
                                    sig1=sig1, sig2=sig2, alloc.ratio=c(1,1), nsim=1e3)
ks.test(alt.dist.samples$samples,
        {(n1+n2 - 2) * k /(n1+n2 - k -1)}*rf(n=1e3, k, n1+n2-k-1,
         ncp = {(n1*n2)/(n1+n2)}*as.vector(crossprod(mu1-mu2, solve(sig1, mu1-mu2))) ) )


# Case 3: Alternate hypothesis is true. The mean difference is non-zero,
# and the covariances of the two groups are different
k <- 5
mu1  <- rep(0,k); del  <- 0.25; mu2 <- mu1 + rep(del, k);
sig1 <- diag(k); sig2 <- sig1 + del*toeplitz(c(1,rep(0.5, k-1)))
alt.dist.samples <- Sim_HotellingT_unequal_var(total_sample_size=200, mean_diff=mu1-mu2,
sig1=sig1, sig2=sig2, alloc.ratio=c(1,1), nsim=1e3)

# Generate samples with unequal allocation ratio:
k <- 8
mu1  <- rep(0,k); del  <- 0.4; mu2 <- mu1 + rep(del, k);
sig1 <- diag(k); sig2 <- sig1 + del*toeplitz(c(1,rep(0.5, k-1)))
alt.dist.samples <- Sim_HotellingT_unequal_var(total_sample_size=150, mean_diff=mu1-mu2,
sig1=sig1, sig2=sig2, alloc.ratio=c(2,1), nsim=1e3)
```

Sum_of_Wishart_df *The approximate degrees of freedom formula for sum of Wishart.*

Description

[Stable]

The approximate degrees of freedom formula for sum of two independent Wishart random variable with parameter sig1 and sig2, and degrees of freedom n1-1 and n2-1 where n1 + n2 is equal to the `total_sample_size`.

See Koner and Luo (2023) for more details on the formula for degrees of freedom.

Usage

```
Sum_of_Wishart_df(total_sample_size, alloc.ratio, sig1, sig2)
```

Arguments

`total_sample_size`	Target sample size, must be a positive integer.
`alloc.ratio`	Allocation of total sample size into the two groups. Must set as a vector of two positive numbers. For equal allocation it should be put as c(1,1), for non-equal allocation one can put c(2,1) or c(3,1) etc.
`sig1`	The true (or estimate) of covariance matrix for the first group. Must be symmetric (`is.symmetric(sig1) == TRUE`) and positive definite (`chol(sig1)` without an error!).
`sig2`	The true (or estimate) of covariance matrix for the second group. Must be symmetric (`is.symmetric(sig2) == TRUE`) and positive definite (`chol(sig2)` without an error!).

Value

The approximate degrees of freedom.

Author(s)

Salil Koner
Maintainer: Salil Koner <salil.koner@duke.edu>

See Also

`Sim_HotellingT_unequal_var()` and `pHotellingT()`.

Examples

```
k <- 8
mu1  <- rep(0,k); del  <- 0.4; mu2 <- mu1 + rep(del, k);
sig1 <- diag(k); sig2 <- sig1 + del*toeplitz(c(1,rep(0.5, k-1)))
alt.dist.samples <- Sum_of_Wishart_df(total_sample_size=150,
sig1=sig1, sig2=sig2, alloc.ratio=c(2,1))
```

Index