Package 'psp'

May 9, 2026

Title Parameter Space Partitioning MCMC for Global Model Evaluation

Version 1.0.5

Date 2026-01-15

Maintainer Lenard Dome <lenarddome@gmail.com>

Description Implements an n-dimensional parameter space partitioning algorithm for evaluating the global behaviour of formal computational models as described by Pitt, Kim, Navarro and Myung (2006) <doi:10.1037/0033-295X.113.1.57>.

License GPL (>= 3)

URL https://github.com/lenarddome/psp

BugReports https://github.com/lenarddome/psp/issues

Imports Rcpp (>= 1.0.8.3), parallel, data.table, methods

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Lenard Dome [aut, cre],
Andy Wills [aut]

Repository CRAN

Date/Publication 2026-01-15 10:30:02 UTC

Contents

psp-package *Parameter Space Partitioning MCMC for Global Model Evaluation*

Description

Implements an n-dimensional parameter space partitioning algorithm for evaluating the global behaviour of formal computational models as described by Pitt, Kim, Navarro and Myung (2006) <doi:10.1037/0033-295X.113.1.57>.

Please cite the package in publications. Use `citation("psp")`.

Author(s)

Lenard Dome

Maintainer: Lenard Dome <lenarddome@gmail.com>

References

Pitt, M. A., Kim, W., Navarro, D. J., & Myung, J. I. (2006). Global model analysis by parameter space partitioning. Psychological Review, 113(1), 57.

pspGlobal *Parameter Space Partitioning*

Description

An all-purpose C++ implementation of the Parameter Space Partitioning MCMC Algorithm described by Pitt, Kim, Navarro, Myung (2006).

Usage

```
pspGlobal(model, discretize, control, save = FALSE, path = ".",
          extension = ".csv", quiet = FALSE)
```

Arguments

`model`	It should take a numeric vector (parameter set) as its argument, and return a numeric vector of continuous variables.
`discretize`	The inequality matrix constructor. It should take a numeric vector of probabilities. It must return a matrix in a `matrix` format with 'type=double'. NA values are not allowed, see Note 1.
`control`	A `list()` of control arguments that tunes the behavior of the parameter space partitioning routine. See Details for more information on what to include.
`save`	if `save = TRUE`, all evaluated parameters and continuous model outputs will be saved to disk. The default is `FALSE`.

`path`	If 'save = TRUE', the path to the file that will store all evaluated parameters and continuous model outputs. The default path is the current working directory. Evaluated parameters and continuous model outputs are save separately, see Details.
`extension`	If 'save = TRUE', the extension of the file will store all evaluated parameters and continuous model outputs. The default extension is `.csv`.
`quiet`	If `FALSE` (default), print the number of the current iteration. If `TRUE`, do not print anything.

Details

Overview:

This function implements the Parameter Space Partitioning algorithm described by Pitt et al. (2006). The brief overview of the algorithm is as follows:

0. Initialize parameter space.

0. Select the first set of parameters, and evaluate the model on this set. Its ordinal output will become the first ordinal pattern and the first region in the parameter space.

1. Pick a random jumping distribution for each ordinal pattern from the sampling region defined by a hypersphere with a center of the last recorded parameter set for a given pattern. Clamp parameter values with their respective lower and upper bounds.

2. Evaluate the model on all new parameter sets.

3. Record new patterns and their corresponding parameter sets. If the parameter sets return an already discovered pattern, add the parameter set to their records. Return to Step 1.

Tuning the behaviour of the algorithm via `control`:

This behavior is further tuned by 'control', which needs to contain a list of the following values:

- *population*. The number of parameter sets in each ordinal region, which serves as a threshold above which pspGlobal will not generate a new jumping distribution for a given ordinal pattern. It has to be an integer.
- *iterations*. The number of global iterations. It has to be an integer. If *population* is not set or the regions have a population less than the upper bound on their size, the function will stop after the set number of *iterations*. If *population* is set, the function will stop after the set number of *iterations* or when all regions have a population greater than or equal to *population*, whichever comes first.
- *lower, upper*. Vectors specifying the lower and upper boundaries of the parameter space for each parameter. The i-th element of lower and upper bounds applies to the i-th parameter. If the parameter is not bounded, set the lower and upper bound to -Inf and Inf respectively.
- *init*. A matrix of parameters to use as the first jumping distribution. Each row contains the parameter set, whereas columns correspond to freely varying parameters of the model. The number of columns must be equal to the number of parameters in the model. The number of rows is arbitrary.
- *radius*. The radius of the hypersphere with n-dimensions to sample from. Must be of type double. If you are unsure what to set here, set it to 1.
- *parameter_names*. A character vector that includes the names of each parameter. The order of elements should correspond to the order of parameter columns in init.

- *stimuli_names*. A character vector that includes the names of each continuous model output. The order of elements should correspond to the order of continuous model output in the mode function.
- *dimensionality*. A single integer that specifies the number of dimensions for the inequality matrix. The inequality matrix is a strict upper triangular matrix. The number of rows and columns is equal to each other.
- *responses*. It is an integer that specifies the number of continuous variables in the model output before the ordinal function is applied. See Note 2.

Saving files to disk:

The evaluated parameter sets and their corresponding continuous model outputs are saved to disk if `save = TRUE`. The evaluated parameter sets are saved in a file with the name `path_parameters` and the extension specified, whereas continuous model outputs are saved in a file with the name `path_continuous` and the extension specified.

Value

The output is a list with the following items:

`ordinal_patterns`
: A 3D array with the ordinal patterns found. The place of the ordinal pattern corresponds to ordinal_counts.

`ordinal_counts`
: A table with the ordinal patterns discovered and the population of their corresponding region - the number of parameter sets discovered to produce the ordinal pattern.

`iterations`
: Number of iterations completed before reaching a set threshold.

Note

1. NA values are usually a result of some parameter combination falling outside of what the model implementation can handle. It is best handled outside of the PSP routine, e.g. during the inequality matrix construction. For example, if NA is detected in the matrix, change all values to 99 before returning the output. 2. Ideally, responses and dimensionality should be the same, but we can imagine a scenario where the dimensionality of the inequality matrix will be smaller than the number of responses. For example, when continuous variables are compressed into a more compact format via clustering.

References

Dome, L., Wills, A. J. (2023) g-distance: On the comparison of model and human heterogeneity. PsyArxiv. doi:10.31234/osf.io/ygmcj.

Pitt, M. A., Kim, W., Navarro, D. J., & Myung, J. I. (2006). Global model analysis by parameter space partitioning. Psychological Review, 113(1), 57. doi:10.1037/0033295X.113.1.57.

Weisstein, Eric W. "Hypersphere Point Picking." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/HyperspherePointPicking.html. Accessed 2021-09-30.

psp_control *Control the behaviour of the psp_global implementation*

Description

psp_control allows users to define characteristics of the parameter space partitioning MCMC algorithm as implemented in psp_global.

Usage

```
psp_control(radius = 0.1, init, lower, upper,
            pop = 400, cl = NULL,
            param_names = NULL,
            parallel = FALSE,
            cluster_names = NULL,
            export_objects = NULL,
            export_libs = NULL,
            iterations = 1000)
```

Arguments

radius	The radius of the hypershere with n-dimensions to sample from. Must be a double or a numeric vector, where elements correspond to parameters in 'init, lower, upper'. Default is 0.1.
init	A vector of parameters to use as the first jumping distribution.
lower, upper	Vectors specifiying the lower and upper boundaries of the parameter space for each parameter. The i-th element of lower and upper bounds applies to the i-th parameter.
pop	The minimum population psp_global aims to find for each ordinal pattern discovered. This can stop the parameter search early in case the population of all ordinal pattern are equal to or larger than pop. If you do not want to use this option, set it to NULL or Inf. Default is 400.
parallel	If TRUE, uses the parallel package to run evaluations of jumping distributions for each chain parallel. Default value is FALSE.
cl	If parallel is TRUE, the number of cores to use for makeCluster from the **parallel** package. If null (default), use all cores.
param_names	A character vector that includes the names of each parameter. If NULL (default), a character vector is generated with parameter_1, parameter_2, parameter_3, ...
cluster_names	Maintained for backwards-compatibility. See export_objects below.
export_objects	A character vector that includes all of the objects to be loaded into each cluster. It is handled by parallel::clusterExports. Default is NULL.
export_libs	A character vector that includes all the packages to be loaded into each cluster. It is handled by parallel::clusterExports. Default is NULL.
iterations	The number of global iterations for psp_global. Default is 1000.

Value

Returns a control list suitable for psp_global with the above elements.

Examples

```
# two parameter model
psp_control(lower = rep(0, 2), upper = rep(1, 2), init = rep(0.5, 2),
            radius = rep(0.25, 2), cluster_names = NULL,
            parallel = FALSE, iterations = 500)
```

---

psp_global *Parameter Space Partitioning*

Description

An all-purpose implementation of the Parameter Space Partitioning MCMC Algorithm described by Pitt, Kim, Navarro, Myung (2006).

Usage

```
psp_global(fn, control = psp_control(), ..., quiet = FALSE)
```

Arguments

fn	The ordinal function. It should take a numeric vector (parameter set) as its argument, and return an ordinal response pattern as character (e.g. "A > B"). NA values are not currently allowed.
control	a list of control parameters, see psp_control
...	Additional arguments passed to fn.
quiet	If FALSE (default), print the total number of patterns found up to the current iteration. If TRUE, do not print anything.

Details

This function implements the Parameter Space Partitioning algorithm desribed by Pitt et al. (2006). The algorithm is as follows:

0. Initialize parameter space.

0. Select first set of parameters, and evaluate the model on this set. Its ordinal output will become the first ordinal pattern and the first region in the parameter space.

1. Pick a random jumping distribution from for each ordinal pattern from the sampling region defined by a hypershere with a center of the last recorded parameter set for a given pattern.

2. Evaluate model on all new parameter sets.

3. Record new patterns and their corresponding parameter sets. If the parameter sets returns an already discovered pattern, add parameter set to their records. Return to Step 1.

This process runs can run in parallel for each discovered pattern.

Value

The output of function psp is a member of the S3 class of PSP. A PSP object is a list with the following items:

ps_partitions	A data.table containing coordinates from the parameter space and their corresponding ordinal response pattern output by fn. Columns include (in this order): parameter coordinates, their ordinal pattern output by fn, the global iteration of the MCMC. Each row corresponds with the evaluation of a single set of parameters.
ps_patterns	A table with the ordinal patterns discovered and the population of their corresponding region - the number of parameter sets discovered to produce the ordinal pattern.
ps_ordinal	A list (if ordinal patterns are multidimensional objects) or character vector (if ordinal patterns are strings or other single values) with the ordinal patterns found. The place of the ordinal pattern corresponds to the names in ps_patterns.

References

Pitt, M. A., Kim, W., Navarro, D. J., & Myung, J. I. (2006). Global model analysis by parameter space partitioning. Psychological Review, 113(1), 57.

Weisstein, Eric W. "Hypersphere Point Picking." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/HyperspherePointPicking.html

Examples

```
library(psp)

#' euclidean distance
#'
#' @param a vector coordinate 1
#' @param b vector coordinate 2
#' @return euclidean distance between coordinates
euclidean <- function(a, b) sqrt(sum((a - b)^2))

# define center points for the 10 regions in a two-dimensional space
positions <- NULL
for (i in seq_len(2)) positions <- cbind(positions, sample(500, 10))

#' dummy hypercube model to test the PSP function
#' The model takes in a set of coordinates, calculates its distance from all
#' all of available coordinates, then return closest region number.
#' This model generalizes to n-dimensions
#'
#' @param x a vector of coordinates
#' @return The number of the region as character
#' @examples
#' model(runif(5))
model <- function(par) {
    areas <- NULL
    for (i in seq_along(par)) {
```

```
        range <- c(1, 0)
        if (i %% 2 == 0) {
            range <- c(0, 1)
        }
        areas <- cbind(areas,
                       seq(range[1], range[2], length.out = 500)[positions[,i]])
    }
    dist <- apply(areas, 1, function(x) euclidean(par, x))
    return(as.character(which.min(dist)))
}

# run Parameter Space Partitioning with some default settings
# Here we run the MCMC for 400 iterations, but the partitioning
# will stop if the population of all regions reach 200.
# Note that we have to load our utility function into
# the clusters, because PSPglobal is currently parallelized.
out <- psp_global(model, psp_control(lower = rep(0, 2),
                                     upper = rep(1, 2),
                                     init = rep(0.5, 2),
                                     radius = rep(0.25, 2),
                                     pop = 100,
                                     parallel = FALSE,
                                     iterations = 100))

print(out)
```

Index