# Package ‘SharedAS’

June 23, 2024

**Type** Package

**Title** Shared active subspace for multi-variate vector-valued functions

**Version** 0.1.0

**Author** Khadija Musayeva, Mickael Binois

**Maintainer** Khadija Musayeva <khmusayeva@gmail.com>

**Description**
Implements several baseline methods to compute a shared active subspace for multi-variate vector-valued functions. It also provides the implementation of the method of Zahm, O., P. G. Constantine, C. Prieur, and Y. M. Marzouk (2020). Gradient-based dimension reduction of multivariate vector-valued functions. SIAM Journal on Scientific Computing 42 (1), A534–A558.

**License** GPL-2

**Encoding** UTF-8

**LazyData** true

**Imports** cpc, geigen, ggplot2, mvtnorm, numDeriv, pracma, RRembo, stats, tidyr, dplyr

**RoxygenNote** 7.3.1

## R topics documented:

| | |
|---|---|
| `methods_grad` | *Computes representative vectors for the Jacobians evaluated at data points and returns the matrix of eigenvectors obtained from the SPD matrix build from such vectors.* |

## Description

Computes representative vectors for the Jacobians evaluated at data points and returns the matrix of eigenvectors obtained from the SPD matrix build from such vectors.

## Usage

```
methods_grad(
  grads,
  n,
  m,
  d,
  method = c("AG", "LP", "MCH"),
  is_grad_norm = FALSE
)
```

## Arguments

| | |
|---|---|
| `grads` | Gradient matrix, where the columns are the concatenation of the objective-wise gradients |
| `n` | Sample size |
| `m` | Number of objectives or function outputs |
| `d` | Number of input dimensions |
| `method` | The method to compute a reprentative vector, either<br>• 'AG' computes the average of gradient vectors<br>• 'LP' for each input variable, computes the rank-1 linear projection of the objective-wise derivatives wrt that variable (based on the leading eigenvector of the SPD matrix of these same derivatives).<br>• 'MCH' computes the minimum-norm element of the convex hull computed from the Jacobian |
| `is_grad_norm` | If set to `TRUE`, performs the L2 normalization of the gradients. This is used only for AG and MCH. |

## Value

Matrix of eigenvectors (column vectors)

## Examples

```
fn <- problem_car_side_impact
d <- 7
n <- 1000
X <- matrix(runif(d * n), n)
Y <- t(apply(X, 1, fn))
grads <- t(apply(X, 1, function(x) c(t(numDeriv::jacobian(fn, x = x)))))
methods_grad(grads, n, ncol(Y), d, method = "MCH")
```

---

`methods_SPD` *Returns the matrix of eigenvectors computed from the stepwise estimation of eigenvectors, joint diagonalization based on the FG algorithm, or from the eigendecomposition of the sum of SPD matrices.*

---

## Description

Returns the matrix of eigenvectors computed from the stepwise estimation of eigenvectors, joint diagonalization based on the FG algorithm, or from the eigendecomposition of the sum of SPD matrices.

## Usage

```
methods_SPD(grads, n, m, d, method = c("SSPD", "SEE", "FG"))
```

## Arguments

| | |
|---|---|
| `grads` | Gradient matrix, where the columns are the concatenation of the objective-wise gradients |
| `n` | Sample size |
| `m` | Number of objectives |
| `d` | Number of input dimensions |
| `method` | Name of the SPD method to choose from<br>• 'SSPD' to compute the eigenvectors from the sum of SPD matrices,<br>• 'SEE' to compute the eigenvectors in a stepwise fashion, or<br>• 'FG' to do simultaneous diagonalization using the Flury and Gautschi algorithm. |

## Value

Matrix of eigenvectors (column vectors)

### References


- Flury, B. N. and W. Gautschi (1986). An algorithm for simultaneous orthogonal transformation of several positive definite symmetric matrices to nearly diagonal form. SIAM Journal on Scientific and Statistical Computing 7 (1), 169–184
- Trendafilov, N. T. (2010). Stepwise estimation of common principal components. Computational Statistics & Data Analysis 54 (12), 3446–3457.


### Examples

```
fn <- problem_car_side_impact
d <- 7
n <- 1000
X <- matrix(runif(d * n), n)
Y <- t(apply(X, 1, fn))
grads <- t(apply(X, 1, function(x) c(t(numDeriv::jacobian(fn, x = x)))))
methods_SPD(grads, n, ncol(Y), d, method = "FG")
```

---

## min_convex_hull *A recursive algorithm to compute the minimum L2-norm point in a polytope.*

---

### Description

A recursive algorithm to compute the minimum L2-norm point in a polytope.

### Usage

```
min_convex_hull(P)
```

### Arguments

| | |
|---|---|
| `P` | Matrix for the convex hull based on which to compute the minimum L2-norm element |

### Value

Vector representing the minimum L2-norm point of the convex hull of `P`

### References


Sekitani, K. and Y. Yamamoto (1993). A recursive algorithm for finding the minimum norm point in a polytope and a pair of closest points in two polytopes. Mathematical Programming 61, 233–249


### Examples

```
X <- matrix(runif(3 * 10), 10)
min_convex_hull(X)
```

problem_car_side_impact

*Car side-impact problem*

## Description

Car side-impact problem

## Usage

```
problem_car_side_impact(
  x,
  params = list(p = NULL, mp = c(0.345, 0.192, 0, 0), sp = c(0.06, 0.06, 10, 10), fixed =
    TRUE)
)
```

## Arguments

| | |
|---|---|
| `x` | a vector in [0, 1]^7 corresponding to 1: Thickness of B-Pillar inner 2: Thickness of B-Pillar reinforcement 3: Thickness of floor side inner 4: Thickness of cross members 5: Thickness of door beam 6 : Thickness of door beltline reinforcement 7 : Thickness of roof rail |
| `params` | List of additional parameters:<br>• `p` optional vector of values for the random parameter, if `fixed = TRUE`, no sampling is done and they are fixed at their mean: 8: Material of B-Pillar inner 9: Material of floor side inner 10: Barrier height 11: Barrier hitting position<br>• `mp,sp` mean and standard deviation for p<br>• `fixed` Logical |

## Value

Vector of size 11

## References

Deb, K. et al. "Reliability-Based Optimization Using Evolutionary Algorithms." Evolutionary Computation, IEEE Transactions on 13.5 (2009): 1054-1074. 2009 Institute of Electrical and Electronics Engineers

## Examples

```
d <- 7
n <- 1000
X <- matrix(runif(d * n), n)
Y <- t(apply(X, 1, problem_car_side_impact))
```

```
pairs(Y)
```

---

`problem_marinedes` *Conceptual marine design problem*

---

## Description

Conceptual marine design problem

## Usage

```
problem_marinedes(x, params = list(constraints = FALSE))
```

## Arguments

| | |
|---|---|
| `x` | Vector in [0,1]^6 |
| `params` | List of additional parameters: <br>• `returnCST`, logical, indicating whether the constraints are returned in the output |

## Value

If returnCST is `TRUE` a vector of size 12, the first 3 are the objectives, next are the constraints. Otherwise, it returns three objectives.

## Note

Adapted from Tanabe, R., & Ishibuchi, H. (2020). An easy-to-use real-world multi-objective optimization problem suite. Applied Soft Computing, 89, 106078.

## References


- M. G. Parsons and R. L. Scott, "Formulation of Multicriterion Design Optimization Problems for Solution With Scalar Numerical Optimization Methods," J. Ship Research, vol. 48, no. 1, pp. 61-76, 2004.
- Tanabe, R., & Ishibuchi, H. (2020). An easy-to-use real-world multi-objective optimization problem suite. Applied Soft Computing, 89, 106078.


## Examples

```
d <- 6
n <- 1000
X <- matrix(runif(d * n), n)
Y <- t(apply(X, 1, problem_marinedes))

pairs(Y[, 1:3])
```

---

`problem_penicillin` *Penicillin production test function based on the implementation github.com/HarryQL/TuRBO-Penicillin.*

---

### Description

Penicillin production test function based on the implementation github.com/HarryQL/TuRBO-Penicillin.

### Usage

```
problem_penicillin(input, params = list(t = 100, returnCST = FALSE))
```

### Arguments

| | |
|---|---|
| `input` | Matrix of input data |
| `params` | List of additional parameters:<br>• `t` Reaction time in hours, defaulted to 100<br>• `returnCST` Optional. If `TRUE` returns the constraints as part of the output. |

### Value

Vector of objective values: if `returnCST` is set to `TRUE` a vector of size 5 where the last 3 columns correspond to the constraints.

### References

Liang, Q. and L. Lai (2021). Scalable bayesian optimization accelerates process optimization of penicillin production. In NeurIPS 2021 AI for Science Workshop.

---

`problem_switch_ripple` *Switching ripple test function. Corresponds to the switching ripple suppressor design problem for voltage source inversion in powered system.*

---

### Description

Switching ripple test function. Corresponds to the switching ripple suppressor design problem for voltage source inversion in powered system.

### Usage

```
problem_switch_ripple(x, params = list(returnCST = FALSE))
```

### Arguments

| | |
|---|---|
| `x` | vector specifying the location where the function is to be evaluated, of size k + 4, k > 1, see Details. |
| `params` | List of additional parameters:<br>• `returnCST` Optional. If set to `TRUE` the last 5 columns in the output correspond to the values of the constraints. |

### Details

Columns of `x` correspond to L1, L2, L3, C1, ..., Ck, Cf where `k` is an arbitrary integer > 1.

Parameters of the problems follow Table 2 in (Zhang et al, 2019).

### Value

Vector of objectives. The first `k` values are related to the suppression of harmonics while the `k+1` one is the sum of inductors.

### References


- Zhang, Z., He, C., Ye, J., Xu, J., & Pan, L. (2019). Switching ripple suppressor design of the grid-connected inverters: A perspective of many-objective optimization with constraints handling. Swarm and evolutionary computation, 44, 293-303.
- He, C., Tian, Y., Wang, H., & Jin, Y. (2019). A repository of real-world datasets for data-driven evolutionary multiobjective optimization. Complex & Intelligent Systems, 1-9.


---

## `problem_synthetic` *Two-dimensional tri-objective problem based on Branin and Currin functions.*

---

### Description

Two-dimensional tri-objective problem based on Branin and Currin functions.

### Usage

```
problem_synthetic(x, params = list(A = NULL, B = NULL, seed = 19))
```

### Arguments

| | |
|---|---|
| `x` | Vector or matrix to evaluate at, takes values in [0,1]^3 |
| `params` | List of additional parameters:<br>• `A` and `B` embedding matrices of size 3x3<br>• seed random seed for reproducibility |

### Note

Problem generated by sampling 2 random matrices and extracting an orthonormal basis from them.

SharedAS *Computes shared active subspace for vector-valued functions.*

## Description

Computes shared active subspace for vector-valued functions.

## Usage

```
SharedAS(
  fn,
  grads = NULL,
  X = NULL,
  distr = list(name = "norm", n = 1000),
  input_dim,
  num_obj,
  start_dim = NULL,
  end_dim = NULL,
  methods = NULL,
  nexps = 1,
  is_norm = TRUE,
  seed = 126,
  params = NULL
)
```

## Arguments

`fn` Vector-valued function for which to compute a shared active subspace

`grads` Optional, matrix of objective-wise gradients concatenated together

`X` Matrix of input data: to provide only if `grads` are provided

`distr` List indicating the data distribution and its parameters:

- `name` either "unif" for the uniform or "norm" for a normal (default is the standard normal) distribution
- `n` number of samples
- `sigma` covariance matrix, if the distribution is set to "norm" (the mean is always zero)

`input_dim` Dimensionality of the input data

`num_obj` Number of outputs of the test function

`start_dim` Optional, start number of the dominating eigenvectors

`end_dim` End number of the dominating eigenvectors: To be set if `start_dim` is set.

`methods` Vector of method names to compute a shared subspace

- AG: average of gradients
- LP: linear projection of gradients

- MCH: minimum convex hull of gradients
- SEE: stepwise estimation of eigenvectors
- SSPD: sum of SPD matrices
- FG: joint diagonalization of SPD matrices
- Zahm: the method of Zahm

`nexps` Number of experiments

`is_norm` Logical, indicates whether to normalize the function outputs in the gradient computation (default is TRUE)

`seed` Random seed for reproducibility

`params` List of additional parameters for the test function

## Details

Depending on the problem, the method of MCH can be costly to compute.

## Value

List of objective-wise root-mean-square error and its sum for each number of dimensions and for each method, over `nexps` number of experiments

## Examples

```
fn <- problem_car_side_impact
input_dim <- 7
num_obj <- 11
nexps <- 5
# distr <- list(name="unif", n=10)
distr <- list(name = "norm", n = 1000)
SharedAS(
  fn = fn, distr = distr, input_dim = input_dim, num_obj = num_obj,
  methods = c("AG", "LP", "MCH", "SSPD", "SEE", "FG", "Zahm"),
  nexps = nexps, is_norm = TRUE, seed = 126, params = NULL
)


fn <- problem_marinedes
input_dim <- 6
num_obj <- 3
nexps <- 1
distr <- list(name = "norm", n = 1000)
SharedAS(
  fn = fn, distr = distr, input_dim = input_dim, num_obj = num_obj,
  methods = c("AG", "LP", "MCH", "SSPD", "SEE", "FG", "Zahm"),
  nexps = nexps, is_norm = TRUE, seed = 126, params = NULL
)


fn <- problem_penicillin
input_dim <- 7
num_obj <- 5
```

```
nexps <- 1
distr <- list(name = "norm", n = 1000)
params <- list(t = 100, returnCST = TRUE)
SharedAS(
  fn = fn, distr = distr, input_dim = input_dim, num_obj = num_obj,
  methods = c("AG", "LP", "MCH", "SSPD", "SEE", "FG", "Zahm"), nexps = nexps,
  is_norm = TRUE, seed = 126, params = params
)


fn <- problem_switch_ripple
input_dim <- 8
num_obj <- 5 # if returCST=TRUE, then numb_obj=10
distr <- list(name = "norm", n = 1000)
nexps <- 1
params <- list(returnCST = FALSE)
SharedAS(
  fn = fn, distr = distr, input_dim = input_dim, num_obj = num_obj,
  methods = c("AG", "LP", "MCH", "SSPD", "SEE", "FG", "Zahm"), nexps = nexps,
  is_norm = TRUE, seed = 126, params = params
)


fn <- problem_synthetic
input_dim <- 3
num_obj <- 2
nexps <- 30
distr <- list(name = "unif", n = 1000)
SharedAS(
  fn = fn, distr = distr, input_dim = input_dim, num_obj = num_obj,
  methods = c("AG", "LP", "MCH", "SSPD", "SEE", "FG", "Zahm"), nexps = nexps,
  is_norm = TRUE, seed = 126
)
```

# Index