

# Package ‘TAD’

January 20, 2025

**Title** Realize the Trait Abundance Distribution

**Version** 1.0.0

**Description** This analytical framework is based on an analysis of the shape of the trait abundance distributions to better understand community assembly processes, and predict community dynamics under environmental changes. This framework mobilized a study of the relationship between the moments describing the shape of the distributions: the skewness and the kurtosis (SKR). The SKR allows the identification of commonalities in the shape of trait distributions across contrasting communities. Derived from the SKR, we developed mathematical parameters that summarise the complex pattern of distributions by assessing (i) the  $R^2$ , (ii) the Y-intercept, (iii) the slope, (iv) the functional stability of community (TADstab), and, (v) the distance from specific distribution families (i.e., the distance from the skew-uniform family a limit to the highest degree of evenness: TADeve).

**License** BSD\_3\_clause + file LICENSE

**URL** [https://forgemia.inra.fr/urep/data\\_processing/tad](https://forgemia.inra.fr/urep/data_processing/tad)

**BugReports** [https://forgemia.inra.fr/urep/data\\_processing/tad/-/issues](https://forgemia.inra.fr/urep/data_processing/tad/-/issues)

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Depends** R ( $\geq 3.5$ )

**Imports** doFuture, foreach, mblm ( $\geq 0.12$ ), methods, stats

**Suggests** Cairo, covr, dplyr, devtools, future ( $\geq 1.33$ ), ggplot2 ( $\geq 3.5$ ), ggpubr ( $\geq 0.6$ ), knitr, Matrix ( $\geq 1.6$ ), pkgdown, rlang, rmarkdown, roxygen2, testthat ( $\geq 3.0$ ), tinytex

**Collate** utils.R saving\_and\_loading\_data.R graph.R tad.R data.R

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Language** en-US

**VignetteBuilder** knitr

**LazyData** true

**LazyDataCompression** bzip2

**NeedsCompilation** no

**Author** Nathan Rondeau [aut],  
 Yoann Le Bagousse-Pinguet [aut]  
 (<<https://orcid.org/0000-0002-5615-5541>>),  
 Raphael Martin [aut] (<<https://orcid.org/0000-0001-8778-7915>>),  
 Lain Pavot [aut, cre],  
 Pierre Liancourt [aut] (<<https://orcid.org/0000-0002-3109-8755>>),  
 Nicolas Gross [aut] (<<https://orcid.org/0000-0001-9730-3240>>),  
 INRAe/UREP [cph]

**Maintainer** Lain Pavot <lain.pavot@inrae.fr>

**Repository** CRAN

**Date/Publication** 2024-11-28 12:20:02 UTC

## Contents

CONSTANTS . . . . .	2
generate_random_matrix . . . . .	3
launch_analysis_tad . . . . .	4
load_abundance_dataframe . . . . .	7
load_statistics_per_obs . . . . .	8
load_statistics_per_random . . . . .	8
load_stat_skr_param . . . . .	9
load_weighted_moments . . . . .	9
moments_graph . . . . .	10
null_model_distribution_stats . . . . .	11
save_abundance_dataframe . . . . .	12
save_statistics_per_obs . . . . .	13
save_statistics_per_random . . . . .	14
save_stat_skr_param . . . . .	14
save_weighted_moments . . . . .	15
skr_graph . . . . .	15
skr_param_graph . . . . .	17
weighted_mvsk . . . . .	19
<b>Index</b>	<b>20</b>

---

CONSTANTS

*The CONSTANTS constant*

---

## Description

Provides a set of constants to prevent typo and provide some defaults values to functions in the TAD.  
 Among those constants are:

- SKEW\_UNIFORM\_SLOPE\_DISTANCE
- SKEW\_UNIFORM\_INTERCEPT\_DISTANCE
- DEFAULT\_SIGNIFICATIVITY\_THRESHOLD

- DEFAULT\_LIN\_MOD
- DEFAULT\_SLOPE\_DISTANCE
- DEFAULT\_INTERCEPT\_DISTANCE

### Usage

CONSTANTS

### Format

An object of class list of length 6.

---

`generate_random_matrix`  
*Generate random matrix*

---

### Description

Generate and save random matrix

### Usage

```
generate_random_matrix(  
  weights,  
  aggregation_factor = NULL,  
  randomization_number,  
  seed = NULL  
)
```

### Arguments

`weights`            the dataframe of weights, one row correspond to a series of observation  
`aggregation_factor`            the dataframe of factor to take into account for the randomization  
`randomization_number`            the number of random abundance matrix to generate  
`seed`                the seed of the pseudo random number generator

### Value

a data.frame of `randomization_number` observations

**Examples**

```

aggregation_factor_name <- c("Year", "Bloc")
weights_factor = TAD::AB[, c("Year", "Plot", "Treatment", "Bloc")]
aggregation_factor <- as.data.frame(
  weights_factor[, aggregation_factor_name]
)
random_matrix <- TAD::generate_random_matrix(
  weights = TAD::AB[, 5:102],
  aggregation_factor = aggregation_factor,
  randomization_number = 100,
  seed = 1312
)
head(random_matrix)

```

---

launch\_analysis\_tad    *Launch the analysis*

---

**Description**

Launch distribution analysis

**Usage**

```

launch_analysis_tad(
  weights,
  weights_factor,
  trait_data,
  randomization_number,
  aggregation_factor_name = NULL,
  statistics_factor_name = NULL,
  seed = NULL,
  abundance_file = NULL,
  weighted_moments_file = NULL,
  stat_per_obs_file = NULL,
  stat_per_rand_file = NULL,
  stat_skr_param_file = NULL,
  regenerate_abundance_df = FALSE,
  regenerate_weighted_moments_df = FALSE,
  regenerate_stat_per_obs_df = FALSE,
  regenerate_stat_per_rand_df = FALSE,
  regenerate_stat_skr_df = FALSE,
  significativity_threshold = CONSTANTS$DEFAULT_SIGNIFICATIVITY_THRESHOLD,
  lin_mod = CONSTANTS$DEFAULT_LIN_MOD,
  slope_distance = CONSTANTS$DEFAULT_SLOPE_DISTANCE,
  intercept_distance = CONSTANTS$DEFAULT_INTERCEPT_DISTANCE,
  csv_tsv_load_parameters = list()
)

```

**Arguments**

weights	the dataframe of weights, one row correspond to a series of observation
weights_factor	the dataframe which contains the different factor linked to the weights
trait_data	a vector of the data linked to the different factor
randomization_number	the number of random abundance matrix to generate
aggregation_factor_name	vector of factor name for the generation of random matrix
statistics_factor_name	vector of factor name for the computation of statistics for each generated matrix
seed	the seed of the pseudo random number generator
abundance_file	the path and name of the RDS file to load/save the dataframe which contains the observed data and the generated matrix
weighted_moments_file	the path and name of the RDS file to load/save the dataframe which contains the calculated moments
stat_per_obs_file	the path and name of the RDS file to load/save the dataframe which contains the statistics for each observed row regarding the random ones
stat_per_rand_file	the path and name of the RDS file to load/save the dataframe which contains the statistics for each random matrix generated
stat_skr_param_file	default=NULL You can provide the output to write the SKR statistics results to.
regenerate_abundance_df	boolean to specify if the abundance dataframe is computed again
regenerate_weighted_moments_df	boolean to specify if the weighted moments dataframe is computed again
regenerate_stat_per_obs_df	boolean to specify if the statistics per observation dataframe is computed again
regenerate_stat_per_rand_df	boolean to specify if the statistics per random matrix dataframe is computed again
regenerate_stat_skr_df	boolean to specify if the stats SKR dataframe is computed again
significativity_threshold	the significance threshold to consider that the observed value is in the randomized value
lin_mod	Indicates the type of linear model to use for (SKR): choose "lm" or "mblm"
slope_distance	slope of the theoretical distribution law (default: slope = 1 intercept = 1.86 skew-uniform distribution family)
intercept_distance	intercept of the theoretical distribution law (default: slope = 1 intercept = 1.86 skew-uniform distribution family)

csv\_tsv\_load\_parameters

a list of parameters for each data structure we want to load. Each element must be named after the data structure we want to load.

### Value

A list of the 9 following named elements:

- raw\_abundance\_df
- filtered\_weights
- filtered\_weights\_factor
- filtered\_trait\_data
- abundance\_df
- weighted\_moments
- statistics\_per\_observation
- stat\_per\_rand
- ses\_skr

### Examples

```
output_path <- file.path(tempdir(), "outputs")
dir.create(output_path)
results <- TAD::launch_analysis_tad(
  weights = TAD::AB[, 5:102],
  weights_factor = TAD::AB[, c("Year", "Plot", "Treatment", "Bloc")],
  trait_data = log(TAD::trait[["SLA"]]),
  aggregation_factor_name = c("Year", "Bloc"),
  statistics_factor_name = (statistics_factor_name <- c("Treatment")),
  regenerate_abundance_df = TRUE,
  regenerate_weighted_moments_df = TRUE,
  regenerate_stat_per_obs_df = TRUE,
  regenerate_stat_per_rand_df = TRUE,
  weighted_moments_file = file.path(output_path, "weighted_moments.csv"),
  stat_per_obs_file = file.path(output_path, "stat_per_obs.csv"),
  stat_per_rand_file = file.path(output_path, "stat_per_rand.csv"),
  stat_skr_param_file = file.path(output_path, "stat_skr_param.csv"),
  randomization_number = 20,
  seed = 1312,
  significativity_threshold = c(0.05, 0.95),
  lin_mod = "lm",
  slope_distance = (
    slope_distance <- TAD::CONSTANTS$SKEW_UNIFORM_SLOPE_DISTANCE
  ),
  intercept_distance = (
    intercept_distance <- TAD::CONSTANTS$SKEW_UNIFORM_INTERCEPT_DISTANCE
  )
)
moments_graph <- TAD::moments_graph(
  moments_df = results$weighted_moments,
  statistics_per_observation = results$statistics_per_observation,
```

```

    statistics_factor_name = statistics_factor_name,
    statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),
    statistics_factor_name_col = c("#1A85FF", "#D41159"),
    output_path = file.path(output_path, "moments_graph.jpeg"),
    dpi = 100
  )
skr_graph <- TAD::skr_graph(
  moments_df = results$weighted_moments,
  statistics_factor_name = statistics_factor_name,
  statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),
  statistics_factor_name_col = c("#1A85FF", "#D41159"),
  output_path = file.path(output_path, "skr_graph.jpeg"),
  slope_distance = slope_distance,
  intercept_distance = intercept_distance,
  dpi = 100
)
skr_param_graph <- TAD::skr_param_graph(
  skr_param = results$ses_skr,
  statistics_factor_name = statistics_factor_name,
  statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),
  statistics_factor_name_col = c("#1A85FF", "#D41159"),
  slope_distance = slope_distance,
  intercept_distance = intercept_distance,
  save_skr_param_graph = file.path(output_path, "skr_param_graph.jpeg"),
  dpi = 100
)

unlink(output_path, recursive = TRUE, force = TRUE)

```

---

load\_abundance\_dataframe

*load\_abundance\_dataframe*

---

## Description

load\_abundance\_dataframe

## Usage

load\_abundance\_dataframe(path, ...)

## Arguments

path	the path to the file to load
...	a set of parameters provided to load_dependent_on_format may contain some operations to apply to format/cast CSV or TSV data which are almost typeless by default

**Value**

an abundance dataframe, with the column number casted into integers and rownames casted into integers.

---

```
load_statistics_per_obs
      load_statistics_per_obs
```

---

**Description**

load\_statistics\_per\_obs

**Usage**

```
load_statistics_per_obs(path, ...)
```

**Arguments**

path	the path to the file to load
...	a set of parameters provided to load_dependent_on_format may contain some operations to apply to format/cast CSV or TSV data which are almost typeless by default

**Value**

a stats par observations dataframe with rownames casted into integers.

---

```
load_statistics_per_random
      load_statistics_per_random
```

---

**Description**

load\_statistics\_per\_random

**Usage**

```
load_statistics_per_random(path, ...)
```

**Arguments**

path	the path to the file to load
...	a set of parameters provided to load_dependent_on_format may contain some operations to apply to format/cast CSV or TSV data which are almost typeless by default



**Value**

a stats per randon dataframe with distance\_to\_family and cv\_distance\_to\_family casted into doubles and with rownames casted into integers.

---

load\_stat\_skr\_param    *load\_stat\_skr\_param*

---

**Description**

load\_stat\_skr\_param

**Usage**

load\_stat\_skr\_param(path, ...)

**Arguments**

path	the path to the file to load
...	a set of parameters provided to load_dependent_on_format may contain some operations to apply to format/cast CSV or TSV data which are almost typeless by default

**Value**

a stats SKR parameters dataframe with distance\_to\_family\_ses and cv\_distance\_to\_family\_ses casted into doubles and with rownames casted into integers.

---

load\_weighted\_moments    *load\_weighted\_moments*

---

**Description**

load\_weighted\_moments

**Usage**

load\_weighted\_moments(path, ...)

**Arguments**

path	the path to the file to load
...	a set of parameters provided to load_dependent_on_format may contain some operations to apply to format/cast CSV or TSV data which are almost typeless by default

**Value**

a weighted moments dataframe with the column number caster into integers and rownames casted into integers.

---

moments_graph	<i>moments_graph</i>
---------------	----------------------

---

**Description**

Graph of the distributions' moments (mean, variance, skewness and kurtosis) compared to null model

**Usage**

```
moments_graph(
  moments_df,
  statistics_per_observation,
  statistics_factor_name,
  statistics_factor_name_breaks = NULL,
  statistics_factor_name_col = NULL,
  output_path = NULL,
  dpi = 600
)
```

**Arguments**

`moments_df` Moments data frame (mean, variance, skewness, kurtosis)

`statistics_per_observation` SES of the Moments data frame and significance compared to null model

`statistics_factor_name` column of data use for colors discrimination

`statistics_factor_name_breaks` vector of factor levels of the `statistics_factor_name`, same dimension than `statistics_factor_name_col`

`statistics_factor_name_col` vector of colors, same dimension than `statistics_factor_name_breaks`

`output_path` The path to save the graph

`dpi` The dpi number to use when we generate png/jpg graph

**Value**

A graph instance

**Examples**

```

results <- TAD::launch_analysis_tad(
  weights = TAD::AB[, 5:102],
  weights_factor = TAD::AB[, c("Year", "Plot", "Treatment", "Bloc")],
  trait_data = log(TAD::trait[["SLA"]]),
  aggregation_factor_name = c("Year", "Bloc"),
  statistics_factor_name = (statistics_factor_name <- c("Treatment")),
  randomization_number = 100
)

# if you want to display the graph
graph <- TAD::moments_graph(
  moments_df = results$weighted_moments,
  statistics_per_observation = results$statistics_per_observation,
  statistics_factor_name = statistics_factor_name,
  statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),
  statistics_factor_name_col = c("#1A85FF", "#D41159")
)

plot(graph)

# if you want to save the graph as a file
# either jpg, jpeg, png or svg are
output_path <- file.path(tempdir(), "outputs")
dir.create(output_path)
TAD::moments_graph(
  moments_df = results$weighted_moments,
  statistics_per_observation = results$statistics_per_observation,
  statistics_factor_name = statistics_factor_name,
  statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),
  statistics_factor_name_col = c("#1A85FF", "#D41159"),
  output_path = file.path(output_path, "moment_graph.png")
)

unlink(output_path, recursive = TRUE, force = TRUE)

```

---

null\_model\_distribution\_stats

*Compare a value to random values*


---

**Description**

Compute different statistics (standardized by the distribution of random values).

**Usage**

```
null_model_distribution_stats(  
  observed_value,  
  random_values,  
  significance_threshold = c(0.05, 0.95),  
  remove_nas = TRUE  
)
```

**Arguments**

`observed_value` the observed value  
`random_values` the random Values  
`significance_threshold`  
the array of values used to compute the quantile (c(0.025, 0.975) by default)  
`remove_nas` boolean - tells whether to remove NAs or not

**Value**

a list corresponding to :

- the observed value
- quantile values (minimum significance threshold)
- quantile values (maximum significance threshold)
- significance (observed value not in quantile values)

**Examples**

```
null_model_distribution_stats(  
  observed_value = 2,  
  random_values = c(1, 4, 5, 6, 8),  
  significance_threshold = c(0.025, 0.975)  
)
```

---

save\_abundance\_dataframe

*save\_abundance\_dataframe*

---

**Description**

This function provides a secured way to save `abundance_dataframe` dataframe. The more generic function provided by TAD `saveDependingOnFormat` expects saves object using their name, but saves nothing if the provided name is not correct, or may even save an unwanted object. This function provides a way to verify the object you want to save, and so, it is more secured.

**Usage**

```
save_abundance_dataframe(path, object = NULL)
```

**Arguments**

path	the path of the file to load
object	the object to save

**Value**

NULL - called for side effects

---

```
save_statistics_per_obs  
    save_statistics_per_obs
```

---

**Description**

This function provides a secured way to save statistics\_per\_obs dataframe. The more generic function provided by TAD `saveDependingOnFormat` expects saves object using their name, but saves nothing if the provided name is not correct, or mya even save an unwanted object. This function provides a way to verify the object you want to save, and so, it is more secured.

**Usage**

```
save_statistics_per_obs(path, object = NULL)
```

**Arguments**

path	the path of the file to load
object	the object to save

**Value**

NULL - called for side effects

---

```
save_statistics_per_random
    save_statistics_per_random
```

---

**Description**

This function provides a secured way to save `statistics_per_random` dataframe. The more generic function provided by TAD `saveDependingOnFormat` expects saves object using their name, but saves nothing if the provided name is not correct, or mya even save an unwanted object. This function provides a way to verify the object you want to save, and so, it is more secured.

**Usage**

```
save_statistics_per_random(path, object = NULL)
```

**Arguments**

<code>path</code>	the path of the file to load
<code>object</code>	the object to save

**Value**

NULL - called for side effects

---

```
save_stat_skr_param    save_stat_skr_param
```

---

**Description**

This function provides a secured way to save `stat_skr_param` dataframe. The more generic function provided by TAD `saveDependingOnFormat` expects saves object using their name, but saves nothing if the provided name is not correct, or mya even save an unwanted object. This function provides a way to verify the object you want to save, and so, it is more secured.

**Usage**

```
save_stat_skr_param(path, object = NULL)
```

**Arguments**

<code>path</code>	the path of the file to load
<code>object</code>	the object to save

**Value**

NULL - called for side effects

---

 save\_weighted\_moments *save\_weighted\_moments*


---

### Description

This function provides a secured way to save weighted\_moments dataframe. The more generic function provided by TAD save\_dependent\_on\_format expects saves object using their name, but saves nothing if the provided name is not correct, or mya even save an unwanted object. This function provides a way to verify the object you want to save, and so, it is more secured.

### Usage

```
save_weighted_moments(path, object = NULL)
```

### Arguments

path	the path of the file to load
object	the object to save

### Value

NULL - called for side effects

---

 skr\_graph *skr\_graph*


---

### Description

Graph of the SKR, compared to null model

### Usage

```
skr_graph(
  moments_df,
  statistics_factor_name,
  statistics_factor_name_breaks = NULL,
  statistics_factor_name_col = NULL,
  slope_distance = CONSTANTS$SKEW_UNIFORM_SLOPE_DISTANCE,
  intercept_distance = CONSTANTS$SKEW_UNIFORM_INTERCEPT_DISTANCE,
  output_path = NULL,
  dpi = 600
)
```

**Arguments**

**moments\_df** moments data frame (mean, variance, skewness, kurtosis)  
**statistics\_factor\_name**  
     column of data use for colors discrimination  
**statistics\_factor\_name\_breaks**  
     vector of factor levels of the statistics\_factor\_name, same dimension than statistics\_factor\_name\_col  
**statistics\_factor\_name\_col**  
     vector of colors, same dimension than statistics\_factor\_name\_breaks  
**slope\_distance** slope of the theoretical distribution law (default: slope = 1 intercept = 1.86 skew-uniform)  
**intercept\_distance**  
     intercept of the theoretical distribution law (default: slope = 1 intercept = 1.86 skew-uniform)  
**output\_path** The path to save the graph  
**dpi** The dpi number to use when we generate png/jpg graph

**Value**

A graph instance

**Examples**

```

results <- TAD::launch_analysis_tad(
  weights = TAD::AB[, 5:102],
  weights_factor = TAD::AB[, c("Year", "Plot", "Treatment", "Bloc")],
  trait_data = log(TAD::trait[["SLA"]]),
  aggregation_factor_name = c("Year", "Bloc"),
  statistics_factor_name = (statistics_factor_name <- c("Treatment")),
  randomization_number = 100,
  slope_distance = (
    slope_distance <- TAD::CONSTANTS$SKEW_UNIFORM_SLOPE_DISTANCE
  ),
  intercept_distance = (
    intercept_distance <- TAD::CONSTANTS$SKEW_UNIFORM_INTERCEPT_DISTANCE
  )
)

graph <- TAD::skr_graph(
  moments_df = results$weighted_moments,
  statistics_factor_name = statistics_factor_name,
  statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),
  statistics_factor_name_col = c("#1A85FF", "#D41159"),
  slope_distance = slope_distance,
  intercept_distance = intercept_distance
)

```



```

plot(graph)

output_path <- file.path(tempdir(), "outputs")
dir.create(output_path)
TAD::skr_graph(
  moments_df = results$weighted_moments,
  statistics_factor_name = statistics_factor_name,
  statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),
  statistics_factor_name_col = c("#1A85FF", "#D41159"),
  slope_distance = slope_distance,
  intercept_distance = intercept_distance,
  dpi = 200,
  output_path = file.path(output_path, "moment_graph.png")
)

unlink(output_path, recursive = TRUE, force = TRUE)

```

---

```

skr_param_graph      skr_param_graph

```

---

## Description

Graph of the parameters computed from the SKR, compared to null model

## Usage

```

skr_param_graph(
  skr_param,
  statistics_factor_name,
  statistics_factor_name_breaks = NULL,
  statistics_factor_name_col = NULL,
  slope_distance = CONSTANTS$SKEW_UNIFORM_SLOPE_DISTANCE,
  intercept_distance = CONSTANTS$SKEW_UNIFORM_INTERCEPT_DISTANCE,
  save_skr_param_graph = NULL,
  dpi = 600
)

```

## Arguments

```

skr_param      SES of SKR parameters data frame (SES and Significance)
statistics_factor_name
                column of data use for colors discrimination
statistics_factor_name_breaks
                vector of factor levels of the statistics_factor_name, same dimension than statistics_factor_name_col

```

**statistics\_factor\_name\_col**  
 vector of colors, same dimension than **statistics\_factor\_name\_breaks**

**slope\_distance** slope of the theoretical distribution law (default: slope = 1 intercept = 1.86 skew-uniform distribution family)

**intercept\_distance**  
 intercept of the theoretical distribution law (default: slope = 1 intercept = 1.86 skew-uniform distribution family)

**save\_skr\_param\_graph**  
 The path to save the graph

**dpi** The dpi number to use when we generate png/jpg graph

### Value

A graph instance

### Examples

```

results <- TAD::launch_analysis_tad(
  weights = TAD::AB[, 5:102],
  weights_factor = TAD::AB[, c("Year", "Plot", "Treatment", "Bloc")],
  trait_data = log(TAD::trait[["SLA"]]),
  aggregation_factor_name = c("Year", "Bloc"),
  statistics_factor_name = (statistics_factor_name <- c("Treatment")),
  randomization_number = 100,
  slope_distance = (
    slope_distance <- TAD::CONSTANTS$SKEW_UNIFORM_SLOPE_DISTANCE
  ),
  intercept_distance = (
    intercept_distance <- TAD::CONSTANTS$SKEW_UNIFORM_INTERCEPT_DISTANCE
  )
)

# if you want to display the graph
graph <- TAD::skr_param_graph(
  skr_param = results$ses_skr,
  statistics_factor_name = statistics_factor_name,
  statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),
  statistics_factor_name_col = c("#1A85FF", "#D41159"),
  slope_distance = slope_distance,
  intercept_distance = intercept_distance
)

plot(graph)

output_path <- file.path(tempdir(), "outputs")
dir.create(output_path)

# if you want to save the graph as a file
# either jpg, jpeg, png or svg are

```

```
TAD::skr_param_graph(  
  skr_param = results$ses_skr,  
  statistics_factor_name = statistics_factor_name,  
  statistics_factor_name_breaks = c("Mown_Unfertilized", "Mown_NPK"),  
  statistics_factor_name_col = c("#1A85FF", "#D41159"),  
  slope_distance = slope_distance,  
  intercept_distance = intercept_distance,  
  save_skr_param_graph = file.path(output_path, "skr_param_graph.jpeg"),  
  dpi = 300  
)  
  
unlink(output_path, recursive = TRUE, force = TRUE)
```

---

weighted\_mvsk

*Compute the weighted mean, variance, skewness and kurtosis*

---

## Description

Compute the weighted mean, variance, skewness and kurtosis of data with given weights

## Usage

```
weighted_mvsk(data, weights)
```

## Arguments

data	the data
weights	the vector or matrix of weights corresponding to the data (each row corresponding to an iteration of data)

## Value

the list of weighted mean, variance, skewness and kurtosis of the data

## Examples

```
weighted_mvsk(  
  data = c(1, 2, 3),  
  weights = matrix(data = c(1, 1, 1, 2, 1, 3), nrow = 2, ncol = 3)  
)
```

# Index

## \* **Statistics**

    null\_model\_distribution\_stats, 11

    weighted\_mvsk, 19

## \* **datasets**

    CONSTANTS, 2

## \* **tad**

    generate\_random\_matrix, 3

    launch\_analysis\_tad, 4

CONSTANTS, 2

generate\_random\_matrix, 3

launch\_analysis\_tad, 4

load\_abundance\_dataframe, 7

load\_stat\_skr\_param, 9

load\_statistics\_per\_obs, 8

load\_statistics\_per\_random, 8

load\_weighted\_moments, 9

moments\_graph, 10

null\_model\_distribution\_stats, 11

save\_abundance\_dataframe, 12

save\_stat\_skr\_param, 14

save\_statistics\_per\_obs, 13

save\_statistics\_per\_random, 14

save\_weighted\_moments, 15

skr\_graph, 15

skr\_param\_graph, 17

weighted\_mvsk, 19