VARS-TOOL

Release 2.3.5

Saman Razavi

Aug 16, 2022

CONTENTS

CHAPTER

ONE

1. OVERVIEW OF VARS-TOOL

VARS-TOOL is a next-generation, multi-method sensitivity and uncertainty analysis software toolbox, applicable to the full range of computer simulation models, including Dynamical Earth Systems Models (DESMs). Developed primarily around the powerful "Variogram Analysis of Response Surfaces" (VARS) framework, VARS-TOOL provides a comprehensive suite of algorithms and methods for global sensitivity analysis (GSA), including ones based on the derivative-based (such as the method of Morris), distribution-based (particularly the variance-based method of Sobol'), and variogram-based (such as VARS) approaches.

The underlying approach of the VARS-TOOL toolkit is described in detail in the following publications:

1. A new framework for comprehensive, robust, and efficient global sensitivity analysis: 1. Theory: https://doi.org/10.1002/2015WR017558
2. A new framework for comprehensive, robust, and efficient global sensitivity analysis: 2. Application: https://doi.org/10.1002/2015WR017559
3. VARS-TOOL: A toolbox for comprehensive, efficient, and robust sensitivity and uncertainty analysis: https://doi.org/10.1016/j.envsoft.2018.10.005
4. Correlation Effects? A Major but Often Neglected Component in Sensitivity and Uncertainty Analysis (GVARS): https://agupubs-onlinelibrary-wiley-com.cyber.usask.ca/doi/full/10.1029/2019WR025436
5. A Fresh Look at Variography: Measuring Dependence and Possible Sensitivities Across Geophysical Systems From Any Given Data (DVARS): https://agupubs-onlinelibrary-wiley-com.cyber.usask.ca/doi/full/10.1029/2020GL089829

CHAPTER

TWO

2. INSTALLATION

2.1 Installing with pip (preferred)

If you have Python3 and pip installed on your machine, then the VARS-TOOL package can be installed as following:

```
foo@bar:~$  pip install varstool
```

2.2 Installing from source code

To install the lastest VARS-TOOL code from the source code, you will need to clone the github repository onto your local device using the command:

```
foo@bar:~$ git clone https://github.com/vars-tool/vars-tool.git
```

To install the package, enter the VARS-TOOL directory and run:

```
foo@bar:~$ cd vars-tool
foo@bar:~$ pip install .
```

If pip is not available on your device use:

```
foo@bar:~$ python setup.py install
```

Note: if installation does not work due to limited permissions, add the `--user` option to the install commands.

CHAPTER

THREE

3. EXAMPLES AND TUTORIALS

You can find examples/tutorials on how to use various aspects of the vars-tool package in the tutorials folder of the github repository. These can be used by cloning the repository to your own device and opening the jupyter notebook tutorials there.

CHAPTER

FOUR

4. VARS-TOOL PACKAGE

4.1 VARS Sensitivity Anlaysis Framework

The Variogram Analysis of Response Surfaces (VARS) is a powerful sensitivity analysis (SA) method first applied to Earth and Environmental System models.

class DVARS(*data_file: str*, *outvarname: str*, *ivars_range: Optional[float] = 0.5*, *phi_max: Optional[float] = 1000000.0*, *phi0: Optional[float] = 1*, *correlation_func_type: Optional[str] = 'linear'*, *tol: Optional[float] = 1e-06*, *report_verbose: Optional[bool] = False*)

Bases: `object`

The Python implementation of the Data Driven Variogram Analysis of Response Surfaces (DVARS) first introduced in Razavi and Gupta (2020) (see [3]_).

simulation_file
[pd.DataFrame] The file name of file containing input and output data.

outvarname
[str] The name of the output variable to calculate sensitivities for. Note that the output variable must be scalar.

Hj
[float, default: 0.5] The fraction of the total parameter space to integrate over. Note that the linear correlation function only has one hyperparameter, so as the Reference notes it is unable to distinguish variogram effects at varying length scales.

tol
[float, default 1e-6] The convergence tolerance for scipy's minimize function acting on the negative log likelihood function.

verbose
[bool, default False] Whether to print diagnostic information.

Razavi, Saman, (2015): algorithm, code in MATALB (c) Blanchard, Cordell, (2022): code in Python 3 Shambaugh, Scott, (2022): code in Python 3

run()
Runs DVARS analysis.

class GVARS(*star_centres: ndarray = array([], dtype=float64)*, *num_stars: Optional[int] = 100*, *parameters: Dict[Union[str, int], Tuple[float, float]] = {}*, *delta_h: Optional[float] = 0.1*, *ivars_scales: Optional[Tuple[float, ...]] = (0.1, 0.3, 0.5)*, *model: Optional[*Model*] = None*, *seed: Optional[int] = 36201460*, *sampler: Optional[str] = None*, *slice_size: Optional[int] = None*, *bootstrap_flag: Optional[bool] = False*, *bootstrap_size: Optional[int] = 1000*, *bootstrap_ci: Optional[float] = 0.9*, *grouping_flag: Optional[bool] = False*, *num_grps: Optional[int] = None*, *report_verbose: Optional[bool] = False*, *corr_mat: ndarray = array([], dtype=float64)*, *num_dir_samples: int = 50*, *fictive_mat_flag: Optional[bool] = True*, *dist_sample_file: Optional[str] = None*)

Bases: *VARS*

The Python implementation of the General Variogram Analysis of Response Surfaces (GVARS), this version can handle correlated factors

Parameters

- **star_centres** (*numpy.array*) – contains star centres of the analysis
- **num_stars** (*int, numpy.int32, numpy.int64, defaults to 100*) – number of stars to generate
- **parameters** (*dict*) – the parameters of the model including lower and upper bounds
- **delta_h** (*float, defaults to 0.1*) – the resolution of star samples
- **ivars_scales** (*tuple, defaults to (0.1, 0.3, 0.5)*) – the IVARS scales
- **model** (varstool.Model) – the model used in the sensitivity analysis
- **seed** (*int, numpy.int32, numpy.int64*) – the seed number used in generating star centres
- **sampler** (*str*) – the type of sampler used to generate star centres
- **slice_size** (*int, numpy.int32, numpy.int64*) – slice size for "plhs" sampler
- **bootstrap_flag** (*bool, defaults to False*) – flag to bootstrap the sensitivity analysis results
- **bootstrap_size** (*int, defaults to 1000*) – the size of bootstrapping experiment
- **bootstrap_ci** (*float, defaults to 0.9*) – the condifence interval of boostrapping
- **grouping_flag** (*bool, defaults to False*) – flag to conduct grouping of sensitive parameters
- **num_grps** (*int, defaults to None*) – the number of groups to categorize parameters
- **report_verbose** (*bool, False*) – flag to show the sensitvity analysis progress
- **corr_mat** (*np.ndarray, np.array([])*) – correlation matrix
- **num_dir_samples** (*int, 50*) – number of directional samples in each star sample
- **fictive_mat_flag** (*bool, False*) – flag that sets if correlation matrix it to be used in place of fictive matrix
- **dist_sample_file** (*str, None*) – file name that contains custom distribution data

Razavi, Saman, (2015): algorithm, code in MATALB (c) Gupta, Hoshin, (2015): algorithm, code in MATLAB (c) Mattot, Shawn, (2019): code in C/++ Do, Nhu, (2020): algorithm, code in MATLAB (c) Keshavarz, Kasra, (2021): code in Python 3 Blanchard, Cordell, (2021): code in Python 3

correlation_plot(*param_names: ndarray*)

plots the correlation between a pair of parameters, displaying the star points and star centres.

Parameters
param_names (*arraylike*) – array containing the names of the two parameters you would like plotted

Returns
ax – the axes of the plot

Return type
matplotlib.axes.Axes

factorimportance_plot(*logy: bool = False*)

plots the ratio of factor importance for IVARS50, VARS-TO, and VARS-ABE

Parameters
logy (*boolean*) – True if variogram plot is to have a logscale y-axis

Returns

- **varax** (*matplotlib.axes.Axes*) – the axes of the variogram plot
- **barfig** (*matplotlib.Figure*) – the figure of the bar chart
- **barax** (*matplotlib.axes.Axes*) – the axes of the bar chart

generate_star() → DataFrame

Generate GVARS star points

Returns
star_points – dataframe containing star points

Return type
pd.DataFrame

run_offline(*model_df*)

runs offline version of GVARS program

Parameters
model_df (*array_like*) – A Pandas Dataframe that contains model ran on generated stars

run_online()

runs online version of GVARS program

property star_centres

returns the star centre samples

property star_points

returns the star point samples.

class Model(*func: Optional[Callable] = None*, *unknown_options: Dict[str, Any] = {}*)

Bases: object

A wrapper class to contain various models and functions to be fed into VARS and its variations. The models can be called by simply calling the wrapper class itself.

Parameters

- **func** (*Callable*) – function of interest
- **unknown_options** (*dict*) – a dictionary of options with keys as parameters and values of parameter values.

class TSGVARS(*star_centres: ndarray = array([], dtype=float64), num_stars: Optional[int] = 100, parameters: Dict[Union[str, int], Tuple[float, float]] = {}, delta_h: Optional[float] = 0.1, ivars_scales: Optional[Tuple[float, ...]] = (0.1, 0.3, 0.5), model: Optional[Model] = None, seed: Optional[int] = 83426192, sampler: Optional[str] = None, slice_size: Optional[int] = None, bootstrap_flag: Optional[bool] = False, bootstrap_size: Optional[int] = 1000, bootstrap_ci: Optional[float] = 0.9, grouping_flag: Optional[bool] = False, num_grps: Optional[int] = None, report_verbose: Optional[bool] = False, corr_mat: ndarray = array([], dtype=float64), num_dir_samples: int = 50, fictive_mat_flag: Optional[bool] = True, dist_sample_file: Optional[str] = None, func_eval_method: Optional[str] = 'serial', vars_eval_method: Optional[str] = 'serial', vars_chunk_size: Optional[int] = None*)

Bases: *GVARS*

Time-series version of GVARS

generate_star() → DataFrame

Generate TSGVARS star points

Returns
star_points – dataframe containing star points

Return type
pd.DataFrame

run_offline(*star_points_eval*, *star_points: Optional = None*)

runs the offline version of TSGVARS program

Parameters

- **star_points_eval** (*array_like*) – A Pandas Dataframe that contains model ran on generated stars
- **star_points** (*array_like*) – A Pandas Dataframe that contains generated stars

run_online()

runs online version of TS-GVARS

class TSVARS(*star_centres=array([], dtype=float64), num_stars: int = 100, parameters: Dict[Union[str, int], Tuple[float, float]] = {}, delta_h: Optional[float] = 0.1, ivars_scales: Optional[Tuple[float, ...]] = (0.1, 0.3, 0.5), model: Optional[Model] = None, seed: Optional[int] = 121430101, sampler: Optional[str] = None, slice_size: Optional[int] = None, bootstrap_flag: Optional[bool] = False, bootstrap_size: Optional[int] = 1000, bootstrap_ci: Optional[int] = 0.9, grouping_flag: Optional[bool] = False, num_grps: Optional[int] = None, report_verbose: Optional[bool] = False, func_eval_method: Optional[str] = 'serial', vars_eval_method: Optional[str] = 'serial', vars_chunk_size: Optional[int] = None*)

Bases: *VARS*

Time-series version of VARS

generate_star() → DataFrame

Generate TSVARS star points

Returns
star_points – dataframe containing star points

Return type
pd.DataFrame

run_offline(*star_points_eval*, *star_points: Optional = None*)

runs the offline version of TSVARS program

Parameters

- **star_points_eval** (*array_like*) – A Pandas Dataframe that contains model ran on generated stars
- **star_points** (*array_like*) – A Pandas Dataframe that contains the generated stars

run_online()

runs online version of TS-VARS

class VARS(*star_centres: ndarray = array([], dtype=float64)*, *num_stars: Optional[int] = 100*, *parameters: Dict[Union[str, int], Tuple[float, float]] = {}*, *delta_h: Optional[float] = 0.1*, *ivars_scales: Optional[Tuple[float, ...]] = (0.1, 0.3, 0.5)*, *model: Optional[Model] = None*, *seed: Optional[int] = 106901406*, *sampler: Optional[str] = None*, *slice_size: Optional[int] = None*, *bootstrap_flag: Optional[bool] = False*, *bootstrap_size: Optional[int] = 1000*, *bootstrap_ci: Optional[float] = 0.9*, *grouping_flag: Optional[bool] = False*, *num_grps: Optional[int] = None*, *report_verbose: Optional[bool] = False*)

Bases: object

The Python implementation of the Variogram Analysis of Response Surfaces (VARS) first introduced in Razavi and Gupta (2015) (see [1]_ and [2]_).

Parameters

- **star_centres** (*numpy.array*) – contains star centres of the analysis
- **num_stars** (*int, numpy.int32, numpy.int64, defaults to 100*) – number of stars to generate
- **parameters** (*dict*) – the parameters of the model including lower and upper bounds
- **delta_h** (*float, defaults to 0.1*) – the resolution of star samples
- **ivars_scales** (*tuple, defaults to (0.1, 0.3, 0.5)*) – the IVARS scales
- **model** (varstool.Model) – the model used in the sensitivity analysis
- **seed** (*int, numpy.int32, numpy.int64*) – the seed number used in generating star centres
- **sampler** (*str*) – the type of sampler used to generate star centres
- **slice_size** (*int, numpy.int32, numpy.int64*) – slice size for "plhs" sampler
- **bootstrap_flag** (*bool, defaults to False*) – flag to bootstrap the sensitivity analysis results
- **bootstrap_size** (*int, defaults to 1000*) – the size of bootstrapping experiment
- **bootstrap_ci** (*float, defaults to 0.9*) – the condifence interval of boostrapping
- **grouping_flag** (*bool, defaults to False*) – flag to conduct grouping of sensitive parameters
- **num_grps** (*int, defaults to None*) – the number of groups to categorize parameters
- **report_verbose** (*bool, False*) – flag to show the sensitvity analysis progress

Razavi, Saman, (2015): algorithm, code in MATALB (c) Gupta, Hoshin, (2015): algorithm, code in MATLAB (c) Mattot, Shawn, (2019): code in C/++ Keshavarz, Kasra, (2021): code in Python 3 Blanchard, Cordell, (2021): code in Python 3

`correlation_plot`(*param_names: ndarray*)

plots the correlation between a pair of parameters, displaying the star points and star centres.

Parameters
param_names (*arraylike*) – array containing the names of the two parameters you would like plotted

Returns
ax – the axes of the plot

Return type
matplotlib.axes.Axes

`factorimportance_plot`(*logy: bool = False*)

plots the ratio of factor importance for IVARS50, VARS-TO, and VARS-ABE

Parameters
logy (*boolean*) – True if variogram plot is to have a logscale y-axis

Returns

- **varax** (*matplotlib.axes.Axes*) – the axes of the variogram plot
- **barfig** (*matplotlib.Figure*) – the figure of the bar chart
- **barax** (*matplotlib.axes.Axes*) – the axes of the bar chart

`generate_star`() → DataFrame

Generate VARS star points

Returns
star_points – dataframe containing star points

Return type
pd.DataFrame

`run_offline`(*model_df*)

runs the offline version of VARS program

Parameters
model_df (*array_like*) – A Pandas Dataframe that contains model ran on generated stars

`run_online`()

runs the online version of the VARS program

`property star_centres`

returns the star centre samples

`property star_points`

returns the star point samples

4.2 Subpackages

4.2.1 varstool.example_models package

4.2.2 varstool.sensitivity_analysis package

4.2.2.1 vars_functions

apply_unique(*func: Callable*, *df: DataFrame*, *axis: int = 1*, *progress: bool = False*) → DataFrame

Apply `func` to unique rows (`axis=1`) or columns (`axis=0`) of `df` in order to increase the efficiency of *func* evaluations.

Parameters

- **func** (*Callable*) – the function of interest to be applied to df
- **df** (*array_like*) – the Pandas DataFrame of interest
- **axis** (*int, optional*) – `0` for *index*, `1` for *columns*, defaults to `1`
- **progress** (*bool, optional*) – `False` for hiding the progress bar, `True` for otherwise, defaults to `False`

Returns
applied_df – the returned dataframe with the *func* evaluations

Return type
array_like

bootstrapping(*num_stars: int*, *pair_df: DataFrame*, *df: DataFrame*, *cov_section_all: DataFrame*, *bootstrap_size: int*, *bootstrap_ci: float*, *delta_h: float*, *ivars_scales: Tuple[float, ...]*, *parameters: Dict[Union[str, int], Tuple[float, float]]*, *st_factor_ranking: DataFrame*, *ivars_factor_ranking: DataFrame*, *grouping_flag: bool*, *num_grps: int*, *progress: bool = False*) → Tuple

performs bootstrapping procedure to gather confidence interval limits on the variogram, Sobol, IVARS and VARS-ABE results, and the reliability estimates of the variogram, Sobol, IVARS, and VARS-ABE results. Also groups the Sobol and IVARS50 results using clustering analysis in a hierarchical order

Parameters

- **num_stars** (*int*) – number of star points
- **pair_df** (*array_like*) – Pandas DataFrame that contains the pairing results of the VARS analysis
- **df** (*array_like*) – Pandas DataFrame containing the star_points and model results
- **cov_section_all** (*array_like*) – Pandas DataFrame containing the sectional covariogram results
- **bootstrap_size** (*int*) – the number of bootstrap samples that were taken
- **bootstrap_ci** (*float*) – the confidence interval of the bootstrapping results (ex. 0.90)
- **delta_h** (*float*) – resolution of star samples
- **ivars_scales** (*tuple*) – tuple containing the scales used in IVARS calculation
- **parameters** (*dictionary*) – dictionary containing parameter names and their attributes
- **st_factor_ranking** (*array_like*) – Pandas DataFrame containing the Sobol factor ranking results

- **ivars_factor_ranking** (*array_like*) – Pandas DataFrame containing the IVARS factor ranking results
- **grouping_flag** (*boolean*) – true if grouping is being done, false otherwise
- **num_grps** (*int*) – the number of groups the parameters are to be clustered into, None if group is to be chosen using elbow method
- **progress** (*boolean*) – true if loading bar is to be shown, false otherwise

Returns

- **gammalb (*array_like*) – variogram upper bound bootstrapping results**
  - **gammaub** (*array_like*) – variogram lower bound bootstrapping results
  - **stlb** (*array_like*) – Sobol lower bound bootstrapping results
  - **stub** (*array_like*) – Sobol upper bound bootstrapping results
  - **ivarslb** (*array_like*) – IVARS lower bound bootstrapping results
  - **ivarsub** (*array_like*) – IVARS upper bound bootstrapping results
  - **rel_sobol_factor_ranking** (*array_like*) – reliability estimates of Sobol results based on bootstrapping
  - **rel_ivars_factor_ranking** (*array_like*) – reliability estimates of IVARS results based on bootstrapping
  - **ivars50_grp** (*array_like*) – IVARS50 grouping results
  - **sobol_grp** (*array_like*) – Sobol grouping results
  - **reli_sobol_grp** (*array_like*) – reliability estimates of groups based on bootstrapping
  - **reli_ivars50_grp** (*array_like*) – reliability estimates of groups based on bootstrapping

References

cov_section(*pair_cols: DataFrame*, *mu_star: DataFrame*) → DataFrame

This function return the sectional covariogram of the pairs of function evaluations that resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

- **pair_cols** (*array_like*) – a Pandas Dataframe of paired values function evaluations
- **mu_star** (*array_like*) – a Pandas DataFrame of mu star values that are calculated separately

Returns

cov_section_values – the sectional covariogram dataframe

Return type

array_like

References

covariogram(*pair_cols: DataFrame*, *mu_overall: Series*) → DataFrame

This function return the covariogram values derived from the pairs of function evaluations that each resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

- **pair_cols** (*array_like*) – a Pandas Dataframe of paired values function evaluations
- **mu_overall** (*array_like*) – a Pandas Dataframe of overall mu calculated on all function evaluation values for each time-step

Returns

covariogram_values – the covariogram dataframe

Return type

array_like

References

e_covariogram(*cov_section_all: DataFrame*) → DataFrame

This function return the Expected value of covariogram values derived from the pairs of function evaluations that each resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

cov_section_all (*array_like*) – a Pandas Dataframe of sectional covariograms

Returns

e_covariogram_values – the covariogram dataframe

Return type

array_like

References

elbow_method(*z: array*) → int

a method used to determine the number of clusters in the data being grouped

Parameters

z (*array_like*) – the results from linking the factors together using statpy's linkage function

Returns

cutoff – the optimal number to use as a cutoff when clustering factors

Return type

int

References

factor_grouping(*sens_idx: DataFrame*, *num_grp: Optional[int] = None*) → DataFrame

Groups parameters based on how close in 'distance' they are. This is done using clustering in a hierarchical fashion. The user can specify the number of groups or have the optimal number chosen using the elbow method by not inputting any group number. Usually done with high parameter models.

Parameters

- **sens_idx** (*array_like*) – the Pandas DataFrame containing the parameters to be grouped along with their values
- **num_grp** (*int*) – the number of groups the parameters are to be clustered into, None if group is to be chosen using elbow method

Returns

- **optm_num_grp** (*int*) – the optimal group number, (either user inputted group number, or calculated group number)
- **rank_grp** (*array_like*) – the group number each parameter belongs to in their corresponding index.
- **clusters** (*array_like*) – list of different cluster configurations used for reliability estimates

References

factor_ranking(*factors: ndarray*) → ndarray

Ranks factors based on their influence (how large or small results are) The lowest rank corresponds to the most influential (larger) factor

Parameters
factors (*array_like*) – an array like object that contains factors/parameters/variables of the sensitivity analysis problem

Returns
ranks – a numpy array containing the ranks of each factor in their corresponding index

Return type
array_like

References

grouping(*result_bs_ivars_df: DataFrame*, *result_bs_sobol: DataFrame*, *result_bs_ivars_ranking: DataFrame*, *result_bs_sobol_ranking: DataFrame*, *num_grps: int*, *st_factor_ranking: DataFrame*, *ivars_factor_ranking: DataFrame*, *parameters: Dict[Union[str, int], Tuple[float, float]]*, *bootstrap_size: int*) → Tuple

Groups parameters based on how close in 'distance' they are. This is done using clustering in a hierarchical fashion. The user can specify the number of groups or have the optimal number chosen using the elbow method by not inputting any group number. Usually done with high parameter models. Also calculates the reliability estimates of the group when bootstrapping.

Parameters

- **result_bs_sobol_ranking** (*array_like*) – Pandas DataFrame with the bootstrapping results of the Sobol factor rankings

- **result_bs_sobol** (*array_like*) – Pandas DataFrame with the bootstrapping results of the Sobol results
- **result_bs_ivars_ranking** (*array_like*) – Pandas DataFrame with the bootstrapping results of the IVARS factor rankings
- **result_bs_ivars_df** (*array_like*) – Pandas DataFrame with the bootstrapping results of the IVARS results
- **num_grps** (*int*) – the number of groups the parameters are to be clustered into, None if group is to be chosen using elbow method
- **st_factor_ranking** (*array_like*) – Pandas DataFrame holding the original Sobol factor rankings
- **ivars_factor_ranking** (*array_like*) – Pandas DataFrame holding the original IVARS factor rankings
- **parameters** (*dictionary*) – dictionary holding the parameter names and their attributes
- **bootstrap_size** (*int*) – the number of bootstrap samples that were taken

Returns

- **ivars50_grp** (*array_like*) – ivars50 grouping results
- **sobol_grp** (*array_like*) – sobol grouping results.
- **reli_sobol_grp** (*array_like*) – reliability estimates of sobol groups using bootstrapping results
- **reli_ivars50_grp** (*array_like*) – relibaility estimates of ivars50 groups using bootstrapping results

References

- ———-
- .. *[1] Razavi, S., & Gupta, H. V. (2016). A new framework for comprehensive,* – robust, and efficient global sensitivity analysis: 1. Theory. Water Resources Research, 52(1), 423-439. doi: 10.1002/2015WR017558
- .. *[2] Razavi, S., & Gupta, H. V. (2016). A new framework for comprehensive,* – robust, and efficient global sensitivity analysis: 1. Application. Water Resources Research, 52(1), 423-439. doi: 10.1002/2015WR017559

ivars(*variogram_array: DataFrame*, *scale: float*, *delta_h: float*) → DataFrame

Generates Integrated Variogram Across a Range of Scales (IVARS) by approximating area using right trapezoids having width of *delta_h* and hights of variogram values. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

- **variogram_array** (*array_like*) – a Pandas Dataframe of variogram values for each time-step
- **scale** (*float*) – the scale for the IVARS evaluations
- **delta_h** (*float*) – the resolution of star point generation

Returns
ivars_values – the Sobol Equivalent values

Return type
array_like

References

morris_eq(*pair_cols: DataFrame*) → DataFrame

This function return the Morris Equivalent values derived from the pairs of function evaluations that each resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters
pair_cols (*array_like*) – a Pandas Dataframe of paired values function evaluations

Returns
morris_eq_values – the morris dataframe

Return type
array_like

References

pairs_h(*iterable: Iterable*) → DataFrame

Give the pairs of numbers considering their differences.

Parameters
iterable (*iterable*) – an iterable object

Returns
pairs – the returned dataframe of paired values

Return type
array_like

scale(*df: DataFrame*, *bounds: DataFrame*, *axis: int = 1*) → DataFrame

This function scales the sampled matrix `df` to the ``bounds`` that is a defined via a dictionary with `ub`, `lb` keys; the values of the dictionary are lists of the upper and lower bounds of the parameters/variables/factors. if (`axis = 1`) then each row of *df* is selected, otherwise columns.

Parameters

- **df** (*array_like*) – a dataframe of randomly sampled values
- **bounds** (*dict*) – a lower and upper bounds to scale the values
- **axis** (*int, optional*) – `0` for index, `1` for columns

Returns
df – the returned dataframe scaled using bounds

Return type
array_like

section_df(*df: DataFrame*, *delta_h: float*) → DataFrame

This function gets the paired values of each section based on index.

Parameters

- **df** (*array_like*) – a dataframe of star points
- **delta_h** (*float*) – resolution of star samples

Returns
sample – the paired values for each section of star points

Return type
array_like

sobol_eq(*gamma: DataFrame*, *ecov: DataFrame*, *variance: Series*, *delta_h: float*) → DataFrame

This function return the Sobol Equivalent values derived from the variogram (*gamma*), expected values of sectional covariograms (*ecov*), and overall variance (*variance*). This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

- **gamma** (*array_like*) – a Pandas Dataframe of variogram values for each time-step
- **ecov** (*array_like*) – a Pandas DataFrame of expected values of sectional covariograms for each time-step
- **variance** (*array_like*) – variance of function evaluations over all time-steps
- **delta_h** (*float*) – resolution of star samples

Returns
sobol_eq_values – the Sobol Equivalent values

Return type
array_like

References

variogram(*pair_cols: DataFrame*) → DataFrame

This function return the variogram calculated from the pairs of function evaluations that each resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters
pair_cols (*array_like*) – a Pandas Dataframe of paired values function evaluations

Returns
variogram_values – the variogram dataframe

Return type
array_like

References

4.2.2.2 gvars_functions

custom_distribution_probabilites(*dist_sample_file: Optional[str]*, *param*)

finds empirical cdf for custom probability distribution and puts it in a dataframe.

Parameters

- **dist_sample_file** (*str*) – string name of .csv file containing custom distribution data
- **param** (*String*) – name of parameter

Returns
cdp – df containing custom distributions and empirical cdf

Return type
array_like

find_boundaries(*parameters*, *dist_sample_file: Optional[str] = None*)

finds maximum and minimum boundary of each parameter.

Parameters

- **parameters** (*Dictionary*) – dictionary containing parameters names and attributes
- **dist_sample_file** (*str*) – name of file containing distributions data

Returns

- **xmin** (*array_like*) – the lower boundaries of each parameter
- **xmax** (*array_like*) – the upper boundaries of each parameter

map_2_cornorm(*parameters: Dict[Union[str, int], Tuple[Union[float, str]]]*, *corr_mat: ndarray*, *progress: bool*) → ndarray

Computes the fictive correlation matrix given a correlation matrix and its corresponding parameters using nataf transformation

Parameters

- **parameters** (*dictionary*) – a dictionary containing parameter names and their attributes
- **corr_mat** (*np.ndarray*) – correlation matrix
- **progress** (*boolean*) – true if loading bar is to be shown, false otherwise

Returns
corr_n – the fictive correlation matrix

Return type
np.ndarray

References

n2x_transform(*norm_vectors: ndarray*, *parameters: Dict*, *dist_sample_file: Optional[str] = None*) → ndarray

transforms multivariate normal samples into parameters original distributions

Parameters

- **norm_vectors** (*np.ndarray*) – multivariate normal samples
- **parameters** (*dict*) – a dictionary containing parameter information (name: bounds, distributions, etc.)
- **dist_sample_file** (*String*) – name of file that contains custom distribution data, optional only for users with custom distributions

Returns
x – the transformed vectors

Return type
np.ndarray

References

pairs_h(*iterable: Iterable*) → DataFrame

Give the pairs of numbers considering their differences.

Parameters
iterable (*iterable*) – an iterable object

Returns
pairs – the returned dataframe of paired values

Return type
array_like

reorder_pairs(*pair_df: DataFrame*, *num_stars: int*, *parameters: Dict[Union[str, int], Tuple[Union[float, str]]]*, *df: DataFrame*, *delta_h: float*, *report_verbose: bool*, *xmax: ndarray*, *xmin: ndarray*, *offline_mode: bool*) → DataFrame

Calculates the differences('h') between the pairings of the star points, and bins and reorders the pair dataframe according to the calculated 'h' values

Parameters

- **pair_df** (*pd.DataFrame*) – Pandas DataFrame containing the paired star points values with the model outputs
- **num_stars** (*int*) – number of star samples
- **parameters** (*dictionary*) – dictionary containing parameter names and their attributes
- **df** (*pd.DataFrame*) – Pandas DataFrame containing the star points
- **delta_h** (*float*) – resolution of star samples
- **report_verbose** (*boolean*) – if True will use a loading bar when generating stars, does nothing if False
- **xmax** (*arraylike*) – array containing max boundary of each parameter
- **xmin** (*arraylike*) – array containing min boundary of each parameter
- **offline_mode** (*boolean*) – if True GVARS analysis is in offline mode, if False it is in online mode

Returns
pair_df – the returned dataframe of paired values

Return type
array_like

rn2rx(*distpair_type: List*, *param1: List*, *param2: List*, *rnpair: float*) → float

transforms value rn in a correlation matrix to values rx

Parameters

- **distpair_type** (*List*) – a list containing parameter 1 and parameter 2's distribution type
- **param1** (*List*) – a list containing statistical information about parameter 1
- **param2** (*List*) – a list containing statistical information about parameter 2
- **rnpair** (*float*) – value containing rn from the correlation matrix

Returns
rx – the transformed rn value

Return type
float

References

rx2rn(*distpair_type: List*, *param1: List*, *param2: List*, *rxpair: float*) → float

transforms value rx in a correlation matrix to value rn

Parameters

- **distpair_type** (*List*) – a list containing parameter 1 and parameter 2's distribution types
- **param1** (*List*) – a list containing statistical information about parameter 1
- **param2** (*List*) – a list containing statistical information about parameter 2
- **rxpair** (*float*) – value containing rx from the correlation matrix

Returns
rn – the transformed rx value

Return type
float

References

4.2.2.3 tsvars_functions

cov_section(*pair_cols: DataFrame*, *mu_star: DataFrame*) → Series

Returns the sectional covariogram of the pairs of function evaluations that resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

- **pair_cols** (*array_like*) – a Pandas Dataframe of paired values function evaluations
- **mu_star** (*array_like*) – a Pandas DataFrame of mu star values that are calculated separately

Returns
cov_section_values – the sectional covariogram dataframe

Return type
array_like

References

covariogram(*pair_cols: DataFrame*, *mu_overall: Series*) → Series

Return the covariogram values derived from the pairs of function evaluations that each resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

- **pair_cols** (*array_like*) – a Pandas Dataframe of paired values function evaluations
- **mu_overall** (*array_like*) – a Pandas Dataframe of overall mu calculated on all function evaluation values for each time-step

Returns
covariogram_values – the covariogram dataframe

Return type
array_like

References

e_covariogram(*cov_section_all: DataFrame*) → Series

Returns the Expected value of covariogram values derived from the pairs of function evaluations that each resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters
cov_section_all (*array_like*) – a Pandas Dataframe of sectional covariograms

Returns
e_covariogram_values – the covariogram dataframe

Return type
array_like

References

ivars(*variogram_array: DataFrame*, *scale: float*, *delta_h: float*) → DataFrame

Generates Integrated Variogram Across a Range of Scales (IVARS) by approximating area using right trapezoids having width of *delta_h* and hights of variogram values. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

- **variogram_array** (*array_like*) – a Pandas Dataframe of variogram values for each time-step
- **scale** (*gloat*) – the scale for the IVARS evaluations
- **delta_h** (*float*) – the resolution of star point generation

Returns
ivars_values – the Sobol Equivalent values

Return type
array_like

References

morris_eq(*pair_cols: DataFrame*) → Tuple[Series, ...]

Return the Morris Equivalent values derived from the pairs of function evaluations that each resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters
pair_cols (*array_like*) – a Pandas Dataframe of paired values function evaluations

Returns
morris_eq_values – the morris dataframe

Return type
array_like

References

pairs_h(*iterable*) → DataFrame

Gives the pairs of numbers considering their differences.

Parameters
iterable (*iterable*) – an iterable object

Returns
pairs – the returned dataframe of paired values

Return type
array_like

scale(*df: DataFrame*, *bounds: DataFrame*, *axis: int = 1*) → DataFrame

Scales the sampled matrix `df` to the `bounds` that is a defined via a dictionary with `ub`, `lb` keys; the values of the dictionary are lists of the upper and lower bounds of the parameters/variables/factors. if (`axis = 1`) then each row of *df* is selected, otherwise columns.

Parameters

- **df** (*array_like*) – a dataframe of randomly sampled values
- **bounds** (*dict*) – a lower and upper bounds to scale the values
- **axis** (`0` for index, `1` for columns) –

Returns
df – the returned dataframe scaled using bounds

Return type
array_like

section_df(*df: DataFrame*, *delta_h: float*) → DataFrame

Gets the paired values of each section based on index.

Parameters

- **df** (*array_like*) – a dataframe of star points
- **delta_h** (*array_like*) – resolution of star samples

Returns
sample – the paired values for each section of star points

Return type
array_like

sobol_eq(*gamma: DataFrame*, *ecov: DataFrame*, *variance: Series*, *delta_h: float*) → Series

Returns the Sobol Equivalent values derived from the variogram (*gamma*), expected values of sectional covariograms (*ecov*), and overall variance (*variance*). This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters

- **gamma** (*array_like*) – a Pandas Dataframe of variogram values for each time-step
- **ecov** (*array_like*) – a Pandas DataFrame of expected values of sectional covariograms for each time-step
- **variance** (*array_like*) – variance of function evaluations over all time-steps
- **delta_h** (*float*) – resolution of star samples

Returns
sobol_eq_values – the Sobol Equivalent values

Return type
array_like

References

variogram(*pair_cols: DataFrame*) → Series

Returns the variogram calculated from the pairs of function evaluations that each resulted from each star point. This function is specific for the time-series varying/aggregate of the VARS sensitivity analysis.

Parameters
pair_cols (*array_like*) – a Pandas Dataframe of paired values function evaluations

Returns
variogram_values – the variogram dataframe

Return type
array_like

References

dvars_functions

L_runner(*phi: ndarray*, *X: ndarray*, *Y: ndarray*, *verbose: bool*) → float

A wrapper function for calculating the negative log-likelihood cost.

Parameters

- **phi** (*numpy.ndarray*) – The hyperparameters for the covariance function.
- **X** (*numpy.ndarray*) – The state matrix for all the input variables percentiles.
- **Y** (*numpy.ndarray*) – The state matrix for the output variables nums.
- **verbose** (*bool, default False*) – whether to print diagnostic information or not

Returns
L – The negative log-likelihood cost.

Return type
float

calc_Gammaj(*Hj: float*, *phij: float*, *variance: float*, *correlation_func_type: str = 'Linear'*) → float

Calculates the IVARS sensitivity index from a learned covariance function.

This integrates the directional variogram for a specific input variable using trapezoidal integration.

Parameters

- **Hj** (*float*) – The fraction of the total parameter space to integrate over. Note that the linear correlation function only has one hyperparameter, so as the Reference notes it is unable to distinguish variogram effects at varying length scales. So, this should not be set to anything other than 1 in practice.
- **phij** (*float*) – The learned hyperparameter for the covariance function between the output and input variable.
- **variance** (*float*) – The variance of the output variable.

- **correlation_func_type** (*str*) – determines the type of covariance function to use

Returns
Gammaj – The global sensitivity index for this output-input variable pair.

Return type
float

calc_L(*phi: np.ndarray*, *X: np.ndarray*, *Y: np.ndarray*) → float

Calculate the negative log-likelihood cost. Note that this is just-in-time compiled by numba for increased speed.

Parameters

- **phi** (*numpy.ndarray*) – The hyperparameters for the covariance function.
- **X** (*numpy.ndarray*) – The state matrix for all the input variables percentiles.
- **Y** (*numpy.ndarray*) – The state matrix for the output variables nums.

Returns
L – The negative log-likelihood cost.

Return type
float

calc_R(*phi: np.ndarray*, *X: np.ndarray*) → np.ndarray

Calculate the correlation matrix between each of the input states. Note that this is just-in-time compiled by numba for increased speed.

Parameters

- **phi** (*numpy.ndarray*) – The hyperparameters for the covariance function.
- **X** (*numpy.ndarray*) – The state matrix for all the input variables percentiles.

Returns
R – The correlation matrix.

Return type
float

calc_Ruw(*phi: np.ndarray*, *Xu: np.ndarray*, *Xw: np.ndarray*) → float

Calculate the correlation between two input states. Note that this is just-in-time compiled by numba for increased speed.

Parameters

- **phi** (*numpy.ndarray*) – The hyperparameters for the covariance function.
- **Xu** (*numpy.ndarray*) – The u'th input state.
- **Xw** (*numpy.ndarray*) – The w'th input state.

Returns
Ruw – The correlation between the two states.

Return type
float

calc_phi_opt(*simulation_df: DataFrame*, *ninvars: int*, *nobvs: int*, *outvarname: str*, *phi_max: float = 1000000.0*, *phi0: float = 1.0*, *tol: float = 1e-06*, *verbose: bool = False*) → ndarray

Calculate the optimal hyperparameters for the covariance functions between the output variable and each of the input variables via maximum likelihood estimation (MLE). MLE works by minimizing a negative log-likelihood function.

Parameters

- **simulation_df** (*pd.Dataframe*) – The input simulation.
- **ninvars** (*int*) – number of input variables
- **nobvs** (*int*) – number of observations
- **outvarname** (*str*) – The name of the output variable to calculate sensitivities for. Note that the output variable must be scalar.
- **tol** (*float, default 1e-6*) – The convergence tolerance for scipy's minimize function acting on the negative log likelihood function.
- **phi_max** (*float, default 1e6*) – the upper bound value for the optimal phi value
- **phi0** (*float, default 1*) – the value where we start our search for the optimal phi.
- **verbose** (*bool, default False*) – Whether to print diagnostic information.

Returns
phi_opt – The learned hyperparameters for the covariance functions.

Return type
numpy.ndarray

calc_rj(*hj: float*, *phij: float*, *correlation_func_type: Optional[str] = 'linear'*) → float

The covariance function (also called a kernel). We currently use a linear kernel which has a single hyperparameter that must be learned. Note that this is just-in-time compiled by numba for increased speed.

Parameters

- **hj** (*float*) – The distance between two state elements.
- **phij** (*float*) – The hyperparameter for the function.
- **correlation_func_type** (*str*) – determines the type of covariance function to be used

Returns
rj – The covariance.

Return type
float

calc_sensitivities(*simulation_df: pd.DataFrame*, *outvarname: str*, *Hj: float = 0.5*, *phi_max: float = 1000000.0*, *phi0: float = 1.0*, *correlation_func_type: str = 'Linear'*, *tol: float = 1e-06*, *verbose: bool = False*) → tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]

Calculates the global sensitivity indices and ratios for a specific output variable to each of a simulation's input variables.

This implements the D-VARS algorithm as dscribed in Reference [1]_ to calculate global sensitivity indices from a set of given data. The implementation largely follows the Reference's supplementary notes, though Multi-SQP is replaced with scipy's minimize function implementing L-BFGS-B. See also Reference [2]_ for some theoretical background on the VARS and IVARS methods of calculating sensitivity indices.

Parameters

- **simulation_df** (*pd.DataFrame*) – The input simulation.
- **outvarname** (*str*) – The name of the output variable to calculate sensitivities for. Note that the output variable must be scalar.
- **Hj** (*float, default: 1.0*) – The fraction of the total parameter space to integrate over. Note that the linear correlation function only has one hyperparameter, so as the Reference

notes it is unable to distinguish variogram effects at varying length scales. So, this should not be set to anything other than 1 in practice.

- **tol** (*float, default 1e-6*) – The convergence tolerance for scipy's minimize function acting on the negative log likelihood function.
- **phi_max** (*float, default 1e6*) – the upper bound value for the optimal phi value
- **phi0** (*float, default 1*) – the value where we start our search for the optimal phi.
- **correlation_func_type** (*str*) – determines the type of covariance function to use
- **verbose** (*bool, default False*) – Whether to print diagnostic information.

Returns

- **sensitivities** (*numpy.ndarray*) – The global sensitivity indices for the output variable for each of the sim's input variables.
- **ratios** (*numpy.ndarray*) – The global sensitivity ratios for the output variable for each of the sim's input variables, essentially the fraction of each input variable's ability to explain the output variance.
- **phi_opt** (*numpy.ndarray*) – the optimal phi values used to calculate the sensitivity indices
- **variance** (*float*) – the variance value of the outvar

References

4.2.3 varstool.sampling package

4.2.3.1 general sampling functions

This module contains 6 different sampling methods, that are: 1. halton sequence 2. lating hypercube sampling (lhs) 3. progressive lating hypercube sampling (plhs) 4. sobol sequence 5. symetrical latin hypercube sampling (symlhs) 6. Generalized Star-Based (gSTAR) Sampling 7. STAR sampling (starvars)

halton(*sp: int*, *params: int*, *seed: Optional[int] = None*, *scramble: bool = True*, *skip: int = 1000*, *leap: int = 101*) → ndarray

Generate quasi-random halton sequence numbers.

This function generates (scrambled) quasi-random halton sequence. In brief, it generalizes the Van der Corput's sequence for multiple dimensions. The Halton sequence uses the base-two Van der Corput sequence for the first dimension, base-three for its second and base-n for its n^{th}-dimension.

Parameters

- **sp** (*int*) – the number of sampling points
- **params** (*int*) – the number of parameters/factors/variables
- **seed** (*int or None, optional*) – seed number for randomization, defaults to `None`
- **scramble** (*bool, optional*) – scrambling flag, defaults to `False`
- **skip** (*int, optional*) – the number of points to skip from the beginning of the sequence, defaults to `1000`
- **leap** (*int, optional*) – the interval of picking values, defaults to `101`

Returns

halton_seq – the halton sequence array

Return type
array_like

References

lhs(*sp=None*, *params=1*, *seed=None*, *criterion=None*, *iterations=None*)

Generate a latin-hypercube design.

Parameters

- **n** (*int*) – the number of factors to generate samples for
- **samples** (*int*) – the number of samples to generate for each factor, defaults to n
- **criterion** (*str*) – allowable values are `center` or `c`, `maximin` or `m`, `centermaximin` or `cm`, and `correlation` or `corr`. If no value given, the design is simply randomized.
- **iterations** (*int*) – The number of iterations in the *maximin* and correlations algorithms, default to 5

Returns
H – An n-by-samples design matrix that has been normalized so factor values are uniformly spaced between zero and one.

Return type
array_like

Example

A 3-factor design (defaults to 3 samples):

```
>>> lhs(3)
array([[ 0.40069325,  0.08118402,  0.69763298],
       [ 0.19524568,  0.41383587,  0.29947106],
       [ 0.85341601,  0.75460699,  0.360024  ]])
```

A 4-factor design with 6 samples:

```
>>> lhs(4, samples=6)
array([[ 0.27226812,  0.02811327,  0.62792445,  0.91988196],
       [ 0.76945538,  0.43501682,  0.01107457,  0.09583358],
       [ 0.45702981,  0.76073773,  0.90245401,  0.18773015],
       [ 0.99342115,  0.85814198,  0.16996665,  0.65069309],
       [ 0.63092013,  0.22148567,  0.33616859,  0.36332478],
       [ 0.05276917,  0.5819198 ,  0.67194243,  0.78703262]])
```

A 2-factor design with 5 centered samples:

```
>>> lhs(2, samples=5, criterion='center')
array([[ 0.3,  0.5],
       [ 0.7,  0.9],
       [ 0.1,  0.3],
       [ 0.9,  0.1],
       [ 0.5,  0.7]])
```

A 3-factor design with 4 samples where the minimum distance between all samples has been maximized:

```
>>> lhs(3, samples=4, criterion='maximin')
array([[ 0.02642564,  0.55576963,  0.50261649],
       [ 0.51606589,  0.88933259,  0.34040838],
       [ 0.98431735,  0.0380364 ,  0.01621717],
       [ 0.40414671,  0.33339132,  0.84845707]])
```

A 4-factor design with 5 samples where the samples are as uncorrelated as possible (within 10 iterations):

```
>>> lhs(4, samples=5, criterion='correlate', iterations=10)
```

plhs(*sp: int*, *params: int*, *slices: int*, *seed: Optional[int] = None*, *iterations: int = 10*, *criterion: str = 'maximin'*) → Tuple[ndarray, ndarray]

Optimize SLHS samples based on [1] and [2]

This function is a Progressive Latin Hypercube Sampling (PLHS) using an optimal Sliced Lating Hypercube Sampling design (SLHS) in the frame of a greedy approach.

Parameters

- **sp** (*int*) – number of sampling points
- **params** (*int*) – number of parameters/factors/variables
- **slices** (*int*) – the number of slices
- **seed** (*int, optional*) – seed number for randomization
- **iterations** (*int, optional*) – number of iterations, defaults to `10`
- **criterion** (*str, optional*) – the criterion for assessing the quality of sample points the available options are: 'maximin' and 'correlation', defaults to `'maximin'`

Returns
plhs_sample_x – the final slhs sample array based on `criterion`

Return type
array_like

References

sobol_sequence(*sp: int*, *params: int*, *seed: Optional[int] = None*, *scramble: bool = True*, *skip: int = 1000*, *leap: int = 101*) → ndarray

Sobol' sequences are low-discrepancy, quasi-random numbers. The code is taken from the *scipy dev-1.7* [1].

Parameters

- **sp** (*int*) – the number of sampling points
- **params** (*int*) – the number of parameters/factors/variables
- **seed** (*int, optional*) – randomization seed number, defaults to `None`
- **scramble** (*bool, optional*) – scrambling the produced array, defaults to `True`
- **skip** (*int, optional*) – the number of points to skip, defaults to `1000`
- **leap** (*int, optional*) – the interval of picking values, defaults to `101`

Returns
sobol_seq – the sobol sequence

Return type
array_like

Notes

There are many versions of Sobol' sequences depending on their "direction numbers". This code uses direction numbers from [4]. Hence, the maximum number of dimension is 21201. The direction numbers have been precomputed with search criterion 6 and can be retrieved at https://web.maths.unsw.edu.au/~fkuo/sobol/

References

symlhs(*sp: int*, *params: int*, *seed: Optional[int] = None*, *criterion: str = 'maximin'*, *iterations: int = 10*) → ndarray

Generate symmetrical LHS of `sp` datapoints in the `params`-dimensional hypercube of [0,1]; developed based on [1].

Parameters

- **sp** (*int*) – the number of sampling points
- **params** (*int*) – the number of parameters/variables/factors
- **seed** (*int or None*) – the seed number for randomization, defaults to `None`
- **criterion** (*str, optional*) – method for evaluation of the generated sampled array, options are `'maximin'` and `'correlation'`, defaults to `'maximin'`
- **iterations** (*int, optional*) – number of iterations to get the optimal sampled array, defaults to `10`

Returns
symlhs_sample – the returned symmetrical LHS sampled array

Return type
array_like

References

4.2.3.2 generalized star-based (gSTAR) sampling functions

g_star(*parameters: Dict[Union[str, int], Tuple[Union[float, str]]]*, *star_centres: ndarray*, *seed: int*, *sampler: str*, *slice_size: int*, *num_stars: int*, *corr_mat: ndarray*, *num_dir_samples: int*, *num_factors: int*, *report_verbose: bool*, *fictive_mat_flag: bool*, *dist_sample_file: Optional[str] = None*) → Tuple[Union[DataFrame, Series], ndarray, ndarray]

This function generates a Pandas Dataframe containing ''star_points" based on [3]

Parameters

- **parameters** (*dictionary*) – dictionary containing parameter names, and their attributes
- **seed** (*int*) – the seed number used in generating star points
- **num_stars** (*int*) – number of star samples
- **corr_mat** (*np.array*) – correlation matrix
- **num_dir_samples** (*int*) – number of directional samples per star point
- **num_factors** (*int*) – number of factors/parameters in model

- **report_verbose** (*boolean*) – if True will use a loading bar when generating stars, does nothing if False
- **fictive_mat_flag** (*boolean*) – if False will use correlation matrix as fictive matrix
- **dist_sample_file** (*str*) – file name of file containing custom distribution data

Returns

- **star_points_df** (*array_like*) – Pandas DataFrame containing the GVARS star points
- **x** (*array_like*) – numpy array containing correlated star centres
- **cov_mat** (*array_like*) – numpy array containing fictive correlation matrix

References

4.2.3.3 STAR sampling (starvars) functions

rangef(*start*, *stop*, *step*, *fround=10*)

Yields sequence of numbers from start (inclusive) to stop (inclusive) by step (increment) with rounding set to n digits.

Parameters

- **start** (*float or int*) – start of sequence
- **stop** (*float or int*) – end of sequence
- **step** (*float or int*) – int or float increment (e.g. 1 or 0.001)
- **fround** (*int*) – float rounding, n decimal places, defaults to 5

Yields

- *int* – yielding the next value of the *range* sequence
- *Source*
- ——
- *The code is taken from the following link, thanks to Goran B.*
- **https** (*//stackoverflow.com/a/49059292/5188208*)

star(*star_centres*, *delta_h: float = 0.1*, *parameters: list = []*, *rettype: str = 'dict'*, *precision: int = 10*) → ndarray

STAR sampling algorithm

This function generates `star_points` based on [1] for each sample set (i.e., each row consisting of `star_centres`). `star_centres` are the points along which in each direction the *star_points* are generated. The resolution of sampling is Δh (`delta_h`). This appraoch is a structured sampling straregy; read more in [2] and [3].

Parameters

- **star_centres** (*array_like*) – the 2d array (n, m) containing sample sets, `n` is the number of sample sets and `m` is the number of parameters/factors/ variables
- **delta_h** (*float, optional*) – sampling resolution, defaults to `0.1`
- **parameters** (*list*) – parameter names
- **rettype** (*str, optional*) – `'dict'` or `'dataframe'`, defaults to `'dict'`

- **precision** (*int, optional*) – the number of digits after the precision point, defaults to 10

Returns

star_points – np.array of star points, each element of this 4d array is a 3d np.array with each 2d array containing star points along each parameter/factor/variable.

Return type

array_like

References

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

V

INDEX

V