UQpy Documentation

Release 1.0.0

Michael D. Shields, Dimitris Giovanis

Jun 08, 2018

Contents:

CHAPTER 1

UQpy package

1.1 Submodules

1.2 UQpy.Distributions module

This module contains functionality for all the distribution supported in UQpy.

1.3 UQpy.Reliability module

This module contains functionality for all the reliability methods supported in UQpy.

class UQpy.Reliability.**SubsetSimulation**(*dimension=None*, *samples_init=None*, *nsamples_ss=None*, *p_cond=None*, *pdf_target_type=None*, *pdf_target=None*, *pdf_target_params=None*, *pdf_proposal_type=None*, *pdf_proposal_scale=None*, *algorithm=None*, *model_type=None*, *model_script=None*, *input_script=None*, *output_script=None*)

Perform Subset Simulation.

This class estimates probability of failure for a user-defined model using Subset Simulation

References: S.-K. Au and J. L. Beck, "Estimation of small failure probabilities in high dimensions by subset simulation,"

Probabilistic Eng. Mech., vol. 16, no. 4, pp. 263–277, Oct. 2001.

Input:

Parameters

- **dimension** (*int*) – A scalar value defining the dimension of target density function. Default: 1
- **nsamples_ss** (*int*) – Number of samples to generate in each conditional subset. No Default Value: nsamples_ss must be prescribed
- **p_cond** (*float*) – Conditional probability at each level. Default: p_cond = 0.1
- **algorithm** (*str*) – Algorithm used to generate MCMC samples. Options:

  'MH': Metropolis Hastings Algorithm 'MMH': Component-wise Modified Metropolis Hastings Algorithm 'Stretch': Affine Invariant Ensemble MCMC with stretch moves

  Default: 'MMH'
- **pdf_target_type** (*str*) – Type of target density function for acceptance/rejection in MMH. Not used for MH or Stretch. Options:

  'marginal_pdf': Check acceptance/rejection for a candidate in MMH using the marginal pdf For independent variables only

  'joint_pdf': Check acceptance/rejection for a candidate in MMH using the joint pdf

  Default: 'marginal_pdf'
- **pdf_target** (*function, function list, or str*) – Target density function from which to draw random samples The target joint probability density must be a function, or list of functions, or a string. If type == 'str'

  The assigned string must refer to a custom pdf defined in the file custom_pdf.py in the working directory

  If type == function The function must be defined in the python script calling MCMC

  If dimension > 1 and pdf_target_type='marginal_pdf', the input to pdf_target is a list of size [dimensions x 1] where each item of the list defines a marginal pdf.

  Default: Multivariate normal distribution having zero mean and unit standard deviation
- **pdf_target_params** (*list*) – Parameters of the target pdf
- **pdf_proposal_type** (*str or str list*) – Type of proposal density function for MCMC. Only used with algorithm = 'MH' or 'MMH' Options:

  'Normal' : Normal proposal density 'Uniform' : Uniform proposal density

  Default: 'Uniform' If dimension > 1 and algorithm = 'MMH', this may be input as a list to assign different proposal

  densities to each dimension. Example pdf_proposal_type = ['Normal','Uniform'].

  If dimension > 1, algorithm = 'MMH' and this is input as a string, the proposal densities for all dimensions are set equal to the assigned proposal type.
- **pdf_proposal_scale** – Scale of the proposal distribution If algorithm == 'MH' or 'MMH'

  For pdf_proposal_type = 'Uniform' Proposal is Uniform in [x-pdf_proposal_scale/2, x+pdf_proposal_scale/2]

  For pdf_proposal_type = 'Normal' Proposal is Normal with standard deviation equal to pdf_proposal_scale

If algorithm == 'Stretch'

pdf_proposal_scale sets the scale of the stretch density g(z) = 1/sqrt(z) for z in [1/pdf_proposal_scale, pdf_proposal_scale]

Default value: dimension x 1 list of ones

- **model_type** (*str*) – Define the model as a python file or as a third party software model (e.g. Matlab, Abaqus, etc.) Options: None - Run a third party software model

  'python' - Run a python model. When selected, the python file must contain a class RunPythonModel that takes, as input, samples and dimension and returns quantity of interest (qoi) in in list form where there is one item in the list per sample. Each item in the qoi list may take type the user prefers.

  Default: None

- **model_script** –

  Defines the script (must be either a shell script (.sh) or a python script (.py)) used to call the model.

  This is a user-defined script that must be provided. If model_type = 'python', this must be a python script (.py) having a specified class

  structure. Details on this structure can be found in the UQpy documentation.

- **input_script** –

  Defines the script (must be either a shell script (.sh) or a python script (.py)) that takes samples generated by UQpy from the sample file generated by UQpy (UQpy_run_{0}.txt) and imports them into a usable input file for the third party solver. Details on UQpy_run_{0}.txt can be found in the UQpy documentation.

  If model_type = None, this is a user-defined script that the user must provide. If model_type = 'python', this is not used.

- **output_script** (*str*) –

  (Optional) Defines the script (must be either a shell script (.sh) or python script (.py)) that extracts quantities of interest from third-party output files and saves them to a file (UQpy_eval_{}.txt) that can be read for postprocessing and adaptive sampling methods by UQpy.

  If model_type = None, this is an optional user-defined script. If not provided, all run files and output files will be saved in the folder 'UQpyOut' placed in the current working directory. If provided, the text files UQpy_eval_{}.txt are placed in this directory and all other files are deleted.

  If model_type = 'python', this is not used.

Type model_script: str

Type input_script: str

Output:

Return self.pf Probability of failure estimate

Rtype self.pf float

Return self.cov Coefficient of variation

Rtype self.cov float

1.4 UQpy.RunModel module

This module contains functionality for the run model method supported in UQpy.

class UQpy.RunModel.**RunModel**(*samples=None*, *dimension=None*, *model_type=None*, *model_script=None*, *input_script=None*, *output_script=None*, *cpu=None*)

A class used to run a computational model a specified sample points.

This class takes samples, either passed as a variable or read through a text file, and runs a specified computational model at those sample points. This can be done by either passing variables and running entirely in python or by calling shell scripts that run a third-party software model.

Input: :param samples: The sample values at which the model will be evaluated. Samples can be passed directly as an array

or can be passed through the text file 'UQpy_Samples.txt'. If passing samples via text file, set samples = None or do not set the samples input.

Parameters

- **dimension** (*int*) – The dimension of the random variable whose samples are being passed to the model.
- **model_type** (*str*) – Define the model as a python file or as a third party software model (e.g. Matlab, Abaqus, etc.) Options: None - Run a third party software model

  'python' - Run a python model. When selected, the python file must contain a class RunPythonModel that takes, as input, samples and dimension and returns quantity of interest (qoi) in in list form where there is one item in the list per sample. Each item in the qoi list may take type the user prefers.

  Default: None
- **model_script** –

  Defines the script (must be either a shell script (.sh) or a python script (.py)) used to call the model.

  This is a user-defined script that must be provided. If model_type = 'python', this must be a python script (.py) having a specified class

  structure. Details on this structure can be found in the UQpy documentation.
- **input_script** –

  Defines the script (must be either a shell script (.sh) or a python script (.py)) that takes samples generated by UQpy from the sample file generated by UQpy (UQpy_run_{0}.txt) and imports them into a usable input file for the third party solver. Details on UQpy_run_{0}.txt can be found in the UQpy documentation.

  If model_type = None, this is a user-defined script that the user must provide. If model_type = 'python', this is not used.
- **output_script** (*str*) –

  (Optional) Defines the script (must be either a shell script (.sh) or python script (.py)) that extracts quantities of interest from third-party output files and saves them to a file (UQpy_eval_{}.txt) that can be read for postprocessing and adaptive sampling methods by UQpy.

If model_type = None, this is an optional user-defined script. If not provided, all run files and output files will be saved in the folder 'UQpyOut' placed in the current working directory. If provided, the text files UQpy_eval_{}.txt are placed in this directory and all other files are deleted.

If model_type = 'python', this is not used.

- **cpu** (*int*) – Number of CPUs over which to run the job. UQpy distributes the total number of model evaluations over this number of CPUs Default: 1 - Runs serially

Type model_script: str

Type input_script: str

Output: :return model_eval: An instance of a sub-class that contains the model solutions. Depending on how the model

is run, model_eval is an instance of a different class.

If model_type = 'python', model_eval is an instance of the class RunPythonModel defined in the python model_script.

If model_type = 'None' and cpu <= 1, model_eval is an instance of the class RunSerial If model_type = 'None' and cpu > 1, model_eval is an instance of the class RunParallel Regardless of model_type, model_eval has the following key attributes:

model_eval.samples = Sample values at which the model has been evaluated. model_eval.QOI = Solution of the model at each sample value.

Return type

model_eval: list In general it is a list. The two key attributes of model_eval have the following type:

model_eval.samples = numpy array model_eval.QOI = list

class RunParallel(*samples=None*, *cpu=None*, *model_script=None*, *input_script=None*, *output_script=None*, *dimension=None*)

A subclass of RunModel to run a third-party software model with parallel processing.

Most attributes of this subclass are inhereted from RunModel. The only variable that is not inherited is QOI.

Input: :param samples: Inherited from RunModel. See its documentation. :type samples: ndarray

Parameters

- **dimension** (*int*) – Inherited from RunModel. See its documentation.
- **model_script** – Inherited from RunModel. See its documentation.
- **input_script** – Inherited from RunModel. See its documentation.
- **output_script** (*str*) – Inherited from RunModel. See its documentation.

Type model_script: str

Type input_script: str

Output: :return QOI: List containing the Quantity of Interest from the simulations

Each item in the list corresponds to one simulation

Rtype QOI list Each item in the list may be of arbitrary data type (e.g. int, float, ndarray, etc.)

class RunSerial(*samples=None*, *dimension=None*, *model_script=None*, *input_script=None*, *output_script=None*)

A subclass of RunModel to run a third-party software model serially (without parallel processing).

Most attributes of this subclass are inherited from RunModel. The only variable that is not inherited is QOI.

Input: :param samples: Inherited from RunModel. See its documentation. :type samples: ndarray

Parameters

- **dimension** (*int*) – Inherited from RunModel. See its documentation.
- **model_script** – Inherited from RunModel. See its documentation.
- **input_script** – Inherited from RunModel. See its documentation.
- **output_script** (*str*) – Inherited from RunModel. See its documentation.

Type model_script: str

Type input_script: str

Output: :return QOI: List containing the Quantity of Interest from the simulations

Each item in the list corresponds to one simulation

Rtype QOI list Each item in the list may be of arbitrary data type (e.g. int, float, ndarray, etc.)

1.5 UQpy.SampleMethods module

This module contains functionality for all the sampling methods supported in UQpy.

class UQpy.SampleMethods.**LHS**(*dimension=1*, *icdf=None*, *icdf_params=None*, *lhs_criterion='random'*, *lhs_metric='euclidean'*, *lhs_iter=100*, *nsamples=None*)

Generate samples based on the Latin Hypercube Design.

A class that creates a Latin Hypercube Design for experiments. Firstly, samples on hypercube [0, 1]^n are generated and then translated to the parameter space.

Input:

Parameters

- **dimension** (*int*) – A scalar value defining the dimension of the random variables Default: len(i_cdf)
- **icdf** (*function/string list*) – Inverse cumulative distribution for each random variable. The inverse cdf may be defined as a function, a string, a list of functions, a list of strings, or a

  list of functions and strings

  Each item in the list specifies the distribution of the corresponding random variable. If icdf[i] is a string, the cdf is defined in Distributions.py or custom_dist.py If icdf[i] is a function, the user must define this function in the script and pass it
- **icdf_params** (*list*) – Parameters of the inverse cdf (icdf) Parameters for each random variable are defined as arrays Each item in the list, icdf_params[i], specifies the parameters for the corresponding inverse cdf,

  icdf[i]

- **lhs_criterion** (*str*) – The criterion for generating sample points Options:

  1. 'random' - completely random
  2. 'centered' - points only at the centre
  3. 'maximin' - maximising the minimum distance between points
  4. 'correlate' - minimizing the correlation between the points

  Default: 'random'

- **lhs_metric** (*str*) – The distance metric to use. Supported metrics are 'braycurtis', 'canberra', 'chebyshev', 'cityblock', 'correlation', 'cosine', 'dice', 'euclidean', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'matching', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule'. Default: 'euclidean'

- **lhs_iter** (*int*) – The number of iteration to run. Required only for maximin, correlate and criterion Default: 100

- **nsamples** (*int*) – Number of samples to generate No Default Value: nsamples must be prescribed

Output :return: LHS.samples: Set of LHS samples :rtype: LHS.samples: ndarray

Returns LHS.samplesU01: Set of uniform LHS samples on [0, 1]^dimension

Return type LHS.samplesU01: ndarray

class UQpy.SampleMethods.**MCMC**(*dimension=None*, *pdf_proposal_type=None*, *pdf_proposal_scale=None*, *pdf_target_type=None*, *pdf_target=None*, *pdf_target_params=None*, *algorithm=None*, *jump=None*, *nsamples=None*, *seed=None*, *nburn=None*)

Generate samples from an arbitrary probability density function using Markov Chain Monte Carlo.

This class generates samples from an arbitrary user-specified distribution using Metropolis-Hastings(MH), Modified Metropolis-Hastings, of Affine Invariant Ensemble Sampler with stretch moves.

References: S.-K. Au and J. L. Beck, "Estimation of small failure probabilities in high dimensions by subset simulation,"

Probabilistic Eng. Mech., vol. 16, no. 4, pp. 263–277, Oct. 2001.

10. **Goodman and J. Weare, "Ensemble samplers with affine invariance," Commun. Appl. Math. Comput. Sci., vol. 5,** no. 1, pp. 65–80, 2010.

Input: :param dimension: A scalar value defining the dimension of target density function.

Default: 1

Parameters

- **pdf_proposal_type** (*str or str list*) – Type of proposal density function for MCMC. Only used with algorithm = 'MH' or 'MMH' Options:

  'Normal' : Normal proposal density 'Uniform' : Uniform proposal density

  Default: 'Uniform' If dimension > 1 and algorithm = 'MMH', this may be input as a list to assign different proposal

  densities to each dimension. Example pdf_proposal_type = ['Normal','Uniform'].

If dimension > 1, algorithm = 'MMH' and this is input as a string, the proposal densities for all dimensions are set equal to the assigned proposal type.

- **pdf_proposal_scale** – Scale of the proposal distribution If algorithm == 'MH' or 'MMH'

  For pdf_proposal_type = 'Uniform' Proposal is Uniform in [x-pdf_proposal_scale/2, x+pdf_proposal_scale/2]

  For pdf_proposal_type = 'Normal' Proposal is Normal with standard deviation equal to pdf_proposal_scale

  If algorithm == 'Stretch'

  pdf_proposal_scale sets the scale of the stretch density g(z) = 1/sqrt(z) for z in [1/pdf_proposal_scale, pdf_proposal_scale]

  Default value: dimension x 1 list of ones

- **pdf_target_type** (*str*) – Type of target density function for acceptance/rejection in MMH. Not used for MH or Stretch. Options:

  'marginal_pdf': Check acceptance/rejection for a candidate in MMH using the marginal pdf For independent variables only

  'joint_pdf': Check acceptance/rejection for a candidate in MMH using the joint pdf

  Default: 'marginal_pdf'

- **pdf_target** (*function, function list, or str*) – Target density function from which to draw random samples The target joint probability density must be a function, or list of functions, or a string. If type == 'str'

  The assigned string must refer to a custom pdf defined in the file custom_pdf.py in the working directory

  If type == function The function must be defined in the python script calling MCMC

  If dimension > 1 and pdf_target_type='marginal_pdf', the input to pdf_target is a list of size [dimensions x 1] where each item of the list defines a marginal pdf.

  Default: Multivariate normal distribution having zero mean and unit standard deviation

- **pdf_target_params** (*list*) – Parameters of the target pdf

- **algorithm** (*str*) – Algorithm used to generate random samples. Options:

  'MH': Metropolis Hastings Algorithm 'MMH': Component-wise Modified Metropolis Hastings Algorithm 'Stretch': Affine Invariant Ensemble MCMC with stretch moves

  Default: 'MMH'

- **jump** – Number of samples between accepted states of the Markov chain. Default value: 1 (Accepts every state)

- **nsamples** (*int*) – Number of samples to generate No Default Value: nsamples must be prescribed

- **seed** (*float or numpy array*) – Seed of the Markov chain(s) For 'MH' and 'MMH', this is a single point, defined as a numpy array of dimension (1 x dimension) For 'Stretch', this is a numpy array of dimension N x dimension, where N is the ensemble size Default:

For 'MH' and 'MMH': zeros(1 x dimension) For 'Stretch': No default, this must be specified.

- **nburn** (*int*) – Length of burn-in. Number of samples at the beginning of the chain to discard. This option is only used for the 'MMH' and 'MH' algorithms. Default: nburn = 0

Type jump: int

Output: :return: MCMC.samples: Set of MCMC samples following the target distribution :rtype: MCMC.samples: ndarray

class UQpy.SampleMethods.**MCS**(*dimension=None*, *icdf=None*, *icdf_params=None*, *nsamples=None*)

Perform Monte Carlo sampling (MCS) of independent random variables from a user-specified probability distribution using inverse transform method.

Parameters

- **dimension** (*int*) – A scalar value defining the dimension of the random variables Default: len(icdf)
- **icdf** (*function/string list*) – Inverse cumulative distribution for each random variable. The inverse cdf may be defined as a function, a string, a list of functions, a list of strings, or a

  list of functions and strings

  Each item in the list specifies the distribution of the corresponding random variable. If icdf[i] is a string, the cdf is defined in Distributions.py or custom_dist.py If icdf[i] is a function, the user must define this function in the script and pass it
- **icdf_params** (*list*) – Parameters of the inverse cdf (icdf) Parameters for each random variable are defined as ndarrays Each item in the list, icdf_params[i], specifies the parameters for the corresponding inverse cdf,

  icdf[i]
- **nsamples** (*int*) – Number of samples to generate No Default Value: nsamples must be prescribed

Output: :return: MCS.samples: Set of generated samples :rtype: MCS.samples: ndarray

Returns MCS.samplesU01: Set of uniform samples on [0, 1]^dimension

Return type MCS.samplesU01: ndarray

class UQpy.SampleMethods.**STS**(*dimension=None*, *icdf=None*, *icdf_params=None*, *sts_design=None*, *input_file=None*)

Generate samples from an assigned probability density function using Stratified Sampling.

References: M.D. Shields, K. Teferra, A. Hapij, and R.P. Daddazio, "Refined Stratified Sampling for efficient Monte Carlo based

uncertainty quantification," Reliability Engineering and System Safety, vol. 142, pp. 310-325, 2015.

Input: :param dimension: A scalar value defining the dimension of target density function.

Default: Length of sts_design

Parameters

- **icdf** (*function/string list*) – Inverse cumulative distribution for each random variable. The inverse cdf may be defined as a function, a string, a list of functions, a list of strings, or a

list of functions and strings

Each item in the list specifies the distribution of the corresponding random variable. If icdf[i] is a string, the cdf is defined in Distributions.py or custom_dist.py If icdf[i] is a function, the user must define this function in the script and pass it

- **icdf_params** (*list*) – Parameters of the inverse cdf (icdf) Parameters for each random variable are defined as arrays Each item in the list, icdf_params[i], specifies the parameters for the corresponding inverse cdf,

  i_cdf[i]

- **sts_design** (*int list*) – Specifies the number of strata in each dimension
- **input_file** (*string*) – File path to input file specifying stratum origins and stratum widths Default: None

Output: :return: STS.samples: Set of stratified samples :rtype: STS.samples: ndarray

Returns STS.samplesU01: Set of uniform stratified samples on [0, 1]^dimension

Return type STS.samplesU01: ndarray

Returns STS.strata: Instance of the class SampleMethods.Strata

Return type STS.strata: ndarray

class UQpy.SampleMethods.**Strata**(*n_strata=None*, *input_file=None*, *origins=None*, *widths=None*)

Define a rectilinear stratification of the n-dimensional unit hypercube with N strata.

Input: :param n_strata: A list of dimension n defining the number of strata in each of the n dimensions

Creates an equal stratification with strata widths equal to 1/n_strata The total number of strata, N, is the product of the terms of n_strata Example - n_strata = [2, 3, 2] creates a 3d stratification with: 2 strata in dimension 0 with stratum widths 1/2 3 strata in dimension 1 with stratum widths 1/3 2 strata in dimension 2 with stratum widths 1/2

:type n_strata int list

Parameters **input_file** (*string*) – File path to input file specifying stratum origins and stratum widths Default: None

Output: :return origins: An array of dimension N x n specifying the origins of all strata

The origins of the strata are the coordinates of the stratum orthotope nearest the global origin Example - A 2D stratification with 2 strata in each dimension origins = [[0, 0]

[0, 0.5] [0.5, 0] [0.5, 0.5]]

Rtype origins array

Return widths An array of dimension N x n specifying the widths of all strata in each dimension Example - A 2D stratification with 2 strata in each dimension widths = [[0.5, 0.5]

[0.5, 0.5] [0.5, 0.5] [0.5, 0.5]]

Rtype widths ndarray

Return weights An array of dimension 1 x N containing sample weights. Sample weights are equal to the product of the strata widths (i.e. they are equal to the size of the

strata in the [0, 1]^n space.

Rtype weights ndarray

static fullfact(*levels*)
Create a full-factorial design

Note: This function has been modified from pyDOE, released under BSD License (3-Clause) Copyright (C) 2012 - 2013 - Michael Baudin Copyright (C) 2012 - Maria Christopoulou Copyright (C) 2010 - 2011 - INRIA - Michael Baudin Copyright (C) 2009 - Yann Collette Copyright (C) 2009 - CEA - Jean-Marc Martinez Original source code can be found at: https://pythonhosted.org/pyDOE/# or https://pypi.org/project/pyDOE/ or https://github.com/tisimst/pyDOE/

Input: :param levels: A list of integers that indicate the number of levels of each input design factor. :type levels: list

Output: :return ff: Full-factorial design matrix :rtype ff: ndarray

1.6 UQpy.Surrogates module

This module contains functionality for all the surrogate methods supported in UQpy.

class UQpy.Surrogates.**SROM**(*samples=None*, *cdf_target=None*, *moments=None*, *weights_errors=None*, *weights_distribution=None*, *weights_moments=None*, *weights_correlation=None*, *properties=None*, *cdf_target_params=None*, *correlation=None*)
Stochastic Reduced Order Model(SROM) provide a low-dimensional, discrete approximation of a given random quantity. SROM generates a discrete approximation of continuous random variables. The probabilities/weights are considered to be the parameters for the SROM and they can be obtained by minimizing the error between the marginal distributions, first and second order moments about origin and correlation between random variables. References: M. Grigoriu, "Reduced order models for random functions. Application to stochastic problems",

Applied Mathematical Modelling, Volume 33, Issue 1, Pages 161-175, 2009.

Input: :param samples: An array/list of samples corresponding to each random variables

Parameters

- **cdf_target** (*list str or list function*) – A list of Cumulative distribution functions of random variables
- **cdf_target_params** (*list*) – Parameters of distribution
- **moments** – A list containing first and second order moment about origin of all random variables
- **weights_errors** (*list*) – Weights associated with error in distribution, moments and correlation. Default: weights_errors = [1, 0.2, 0]
- **properties** (*list*) – A list of booleans representing properties, which are required to match in reduce order model. This class focus on reducing errors in distribution, first order moment about origin, second order moment about origin and correlation of samples. Default: properties = [True, True, True, False] Example: properties = [True, True, False, False] will minimize errors in distribution and errors in first order moment about origin in reduce order model.
- **weights_distribution** – An list or array containing weights associated with different samples. Options:

  If weights_distribution is None, then default value is assigned. If size of weights_distribution is 1xd, then it is assigned as dot product

  of weights_distribution and default value.

Otherwise size of weights_distribution should be equal to Nxd.

Default: weights_distribution = Nxd dimensional array with all elements equal to 1.

- **weights_moments** (*ndarray or list (float)*) – An array of dimension 2xd, where 'd' is number of random variables. It contain weights associated with moments. Options:

  If weights_moments is None, then default value is assigned. If size of weights_moments is 1xd, then it is assigned as dot product

  of weights_moments and default value.

  Otherwise size of weights_distribution should be equal to 2xd.

  Default: weights_moments = Square of reciprocal of elements of moments.

- **weights_correlation** – An array of dimension dxd, where 'd' is number of random variables. It contain weights associated with correlation of random variables. Default: weights_correlation = dxd dimensional array with all elements equal to 1.
- **correlation** – Correlation matrix between random variables.

Output: :return: SROM.sample_weights: The probabilities weights for each sample as identified through optimization. :rtype: SROM.sample_weights: ndarray

1.7 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

u

Index