utlvce

Release 0.1.1

Juan L. Gamella

Apr 01, 2022

CONTENTS:

1 How to run the UT-LVCE algorithms **3**

2 Versioning **5**

3 License **7**

4 Feedback **9**

4.1 Running the UT-LVCE algorithms 9
4.2 utlvce.Model 13
4.3 utlvce.score 20
4.4 utlvce.generators 23

Python Module Index **29**

Index **31**

The `utlvce` package is a Python implementation of the UT-LVCE algorithms from the 2022 paper "Perturbations and Causality in Gaussian Latent Variable Models", by A. Taeb, JL. Gamella, C. Heinze-Deml and P. Bühlmann.

You can find the source code in the package's GitHub repository. The code to reproduce the experiments and figures in the paper can be found in a separate repository.

Installation

You can install the latest version of the package using pip, i.e.

```
pip install utlvce
```

CHAPTER

ONE

HOW TO RUN THE UT-LVCE ALGORITHMS

If you're interested in running the algorithms proposed in the paper for your own work, you can find details on how to do so and examples under the section *Running the UT-LVCE algorithms*.

This page also documents the other functionalities of the package, such as generating synthetic data and random models for your own experiments (see `utlvce.generators` and `utlvce.model`). If you're interested in the alternating optimization procedure from section 4.1, you can find more details under `utlvce.score`.

CHAPTER

TWO

VERSIONING

The pacakge is still at its infancy and its API may change in the future. Non backward-compatible changes to the API are reflected by a change to the minor or major version number, e.g.

> *code written using utlvce==0.1.2 will run with utlvce==0.1.3, but may not run with utlvce==0.2.0.*

CHAPTER

THREE

LICENSE

The code is open-source and shared under a BSD 3-Clause License. You can find the full license and the source code in the GitHub repository.

CHAPTER

FOUR

FEEDBACK

Feedback is most welcome! You can add an issue in the repository or send an email.

4.1 Running the UT-LVCE algorithms

The paper proposes two algorithms for estimating the interventional equivalence class, and a full causal discovery procedure:

- Algorithm 1: Estimating the equivalence class of best scoring DAGs from a set of candidate DAGs.
- Algorithm 2: Improving on the initial candidate set of DAGs and estimating the equivalence class of best scoring one.
- UT-LVCE as a Causal Discovery procedure, using the output of GES on the pooled data as an initial candidate set.

These are provided through the functions *utlvce.equivalence_class()* and *utlvce.equivalence_class_w_ges()*, documented below.

We now provide details and examples on how to run the algorithms; the example code can also be found here. To this end we will first generate some synthetic data using the *utlvce.generators* module:

```
import utlvce
import utlvce.generators as generators
import utlvce.utils as utils

# Generate a random model and sample some data
model = generators.random_graph_model(
    20, 2.1, {1, 2, 3}, 2, 5, 0.5, 0.6, 3, 6, 0.2, 0.3, 0.2, 1, 0.7, 0.8)
data = model.sample(1000, random_state=42)

# For algorithm 1 and 2, we will use the Markov Equivalence class of the data generating
# graph as a candidate set
candidate_dags = utils.mec(model.A)
```

Algorithm 1

Algorithm 1 can be run by setting *prune_edges=False* when calling *utlvce.equivalence_class()* with the data and a candidate set of DAGs.

```
# Algorithm 1: Estimate equivalence class of the best scoring
# candidate
utlvce.equivalence_class(candidate_dags, data, prune_edges=False)
```

Algorithm 2

Conversely, algorithm 2 can be run by setting *prune_edges=True* when calling `utlvce.equivalence_class()`.

```
# Algorithm 2: Improve on the initial candidate set and estimate the
# equivalence class of the best scoring graph
utlvce.equivalence_class(candidate_dags, data, prune_edges=True)
```

UT-LVCE as a Causal Discovery Procedure

For the causal discovery procedure, the `utlvce.equivalence_class_w_ges()` function is provided. This will estimate the equivalence class of the data-generating model, using the output of GES on the pooled data as an initial set of candidate DAGs.

Note that the call would be equivalent to passing the output DAGs of GES to `utlvce.equivalence_class()` with *prune_edges=True*. Any other causal discovery procedure can be used to generate a set of initial candidate DAGs, and used together with UT-LVCE in this way.

```
# GES + UT-LVCE: Estimate equivalence class of the data generating
# model using GES to obtain the initial set of candidate graphs
utlvce.equivalence_class_w_ges(data)
```

4.1.1 Full function specification

Both functions can take additional parameters to control, among other things, the behaviour of the alternating optimization procedure used to compute the likelihood of the DAGs. The complete specification follows.

utlvce.**equivalence_class**(*candidate_dags*, *data*, *prune_edges=False*, *nums_latent=[1, 2, 3]*, *folds=[0.5, 0.25, 0.25]*, *psi_max=None*, *psi_fixed=False*, *max_iter=1000*, *threshold_dist_B=0.001*, *threshold_fluctuation=1e-05*, *max_fluctuations=5*, *threshold_score=1e-05*, *learning_rate=1*, *B_solver='grad'*, *random_state=42*, *verbose=0*)

Estimate the equivalence class of the best scoring graph from an initial set of candidate DAGs.

Parameters

- **candidate_dags** (*list of numpy.ndarray*) – A list of the candidate DAG adjacencies, where for each adjacency A, *A[i, j] != 0 implies i -> j*.
- **data** (*list of numpy.ndarray*) – A list with the sample from each environment, where each sample is an array with columns corresponding to variables and rows to observations.
- **prune_edges** (*bool, default=False*) – If we also consider the pruned versions of the candidate graphs.
- **nums_latent** (*list of ints, default=[1,2,3]*) – The candidates for the number of hidden variables, from which one is selected via cross-validation.
- **folds** (*list of float, default=[0.5, 0.25, 0.25]*) – The size of the different splits of the data, where the first corresponds to the training data (used to fit the models), the second is used to select the number of latent variables and best DAG, and the third is used to select the intervention targets.
- **psi_max** (*float, default = None*) – The maximum allowed change in variance between environments for the hidden variables. If None, psis are unconstrained.

- **psi_fixed** (*bool, default = False*) – If *True*, impose the additional constraint that the hidden variables have all the same variance.
- **max_iter** (*int, default=1000*) – The maximum number of iterations allowed for the alternating minimization procedure.
- **threshold_dist_B** (*float, default=1e-3*) – If the change in B between successive iterations of the alternating optimization procedure is lower than this threshold, stop the procedure and return the estimate.
- **threshold_fluctuation** (*float, default=1e-5*) – For the alternating optimization routine, if the score worsens by more than this value, consider the iteration a fluctuation.
- **max_fluctuations** (*int, default=5*) – For the alternating optimization routine, the maximum number of fluctuations(see above) allowed before stopping the subroutine.
- **threshold_score** (*float, default=1e-5*) – For the gradient descent subroutines, if change in score between successive iterations is below this threshold, stop.
- **learning_rate** (*float, default=1*) – The initial learning rate(factor by which gradient is multiplied) for the gradient descent subroutines.
- **B_solver** (*{'grad', 'adaptive', 'cvx'}, default='grad'*) – Sets the solver for the connectivity matrix B, where the options are (ordered by decreasing speed and increasing stability) *grad*, *adaptive* and *cvx*.
- **random_state** (*int, default=42*) – To set the random state for reproducibility when randomly splitting the data. Successive calls with the same random state will have the same result.
- **verbose** (*int, default = 0*) – If debug and execution traces should be printed. *0* corresponds to no traces, higher values correspond to higher verbosity.

Returns

- **estimated_icpdag** (*numpy.ndarray*) – The I-CPDAG representing the estimated equivalence class.
- **estimated_I** (*set of ints*) – The estimated set of intervention targets.
- **estimated_model** (*utlvce.model.Model*) – The estimated model. Its underlying DAG adjacency can be accessed by *estimated_model.A*. The fitted parameters and assumption deviation metrics can be seen by calling *print(estimated_model)* (see *utlvce.model* module).

Raises

- **ValueError :** – If (1) the given data is not valid, i.e. (different number of variables per sample, or one sample with a single observation), (2) if the given *B_solver* is not valid or (3) if one of the candidate adjacencies does not correspond to a DAG.
- **TypeError :** – If the given data is not a list of *numpy.ndarray*.

utlvce.**equivalence_class_w_ges**(*data*, *nums_latent=[1, 2, 3]*, *folds=[0.5, 0.25, 0.25]*, *psi_max=None*, *psi_fixed=False*, *max_iter=1000*, *threshold_dist_B=0.001*, *threshold_fluctuation=1e-05*, *max_fluctuations=5*, *threshold_score=1e-05*, *learning_rate=1*, *B_solver='grad'*, *random_state=42*, *verbose=0*)

Estimate the equivalence class of the data-generating model, using the output of GES on the pooled data as an initial set of candidate DAGs.

Parameters

- **data** (*list of numpy.ndarray*) – A list with the sample from each environment, where each sample is an array with columns corresponding to variables and rows to observations.

- **nums_latent** (*list of ints, default=[1,2,3]*) – The candidates for the number of hidden variables, from which one is selected via cross-validation.
- **folds** (*list of float, default=[0.5, 0.25, 0.25]*) – The size of the different splits of the data, where the first corresponds to the training data (used to fit the models), the second is used to select the number of latent variables and best DAG, and the third is used to select the intervention targets.
- **psi_max** (*float, default = None*) – The maximum allowed change in variance between environments for the hidden variables. If None, psis are unconstrained.
- **psi_fixed** (*bool, default = False*) – If *True*, impose the additional constraint that the hidden variables have all the same variance.
- **max_iter** (*int, default=1000*) – The maximum number of iterations allowed for the alternating minimization procedure.
- **threshold_dist_B** (*float, default=1e-3*) – If the change in B between successive iterations of the alternating optimization procedure is lower than this threshold, stop the procedure and return the estimate.
- **threshold_fluctuation** (*float, default=1e-5*) – For the alternating optimization routine, if the score worsens by more than this value, consider the iteration a fluctuation.
- **max_fluctuations** (*int, default=5*) – For the alternating optimization routine, the maximum number of fluctuations(see above) allowed before stopping the subroutine.
- **threshold_score** (*float, default=1e-5*) – For the gradient descent subroutines, if change in score between successive iterations is below this threshold, stop.
- **learning_rate** (*float, default=1*) – The initial learning rate(factor by which gradient is multiplied) for the gradient descent subroutines.
- **B_solver** (*{'grad', 'adaptive', 'cvx'}, default='grad'*) – Sets the solver for the connectivity matrix B, where the options are (ordered by decreasing speed and increasing stability) *grad*, *adaptive* and *cvx*.
- **random_state** (*int, default=42*) – To set the random state for reproducibility when randomly splitting the data. Successive calls with the same random state will have the same result.
- **verbose** (*int, default = 0*) – If debug and execution traces should be printed. *0* corresponds to no traces, higher values correspond to higher verbosity.

Returns

- **estimated_icpdag** (*numpy.ndarray*) – The I-CPDAG representing the estimated equivalence class.
- **estimated_I** (*set of ints*) – The estimated set of intervention targets.
- **estimated_model** (*utlvce.model.Model*) – The estimated model. Its underlying DAG adjacency can be accessed by *estimated_model.A*. The fitted parameters and assumption deviation metrics can be seen by calling *print(estimated_model)* (see *utlvce.model* module).

Raises

- **ValueError :** – If (1) the given data is not valid, i.e. (different number of variables per sample, or one sample with a single observation), (2) if the given *B_solver* is not valid.
- **TypeError :** – If the given data is not a list of *numpy.ndarray*.

4.2 utlvce.Model

The *utlvce.Model* class is a representation of a linear Gaussian structural causal model with latent effects. It is used throughout the code, e.g. in the alternating optimization procedure implemented in *utlvce.score* and to generate synthetic data (see *utlvce.Model.sample()*).

The class also implements the `__str__` method; calling `print(model)` will return a human-readable representation of the model parameters and the assumption deviation metrics.

class utlvce.**Model**(*A*, *B*, *gamma*, *omegas*, *psis*)

The *utlvce.Model* class holds the parameters of the model and offers additional functionality such as checking deviation from assumptions or generating intermediate quantities used in the alternating optimization procedure. It also allows generating data according to the model (see *sample()* below).

It defines the following parameters:

Parameters

- **p** (*int*) – The number of observed variables in the model.
- **l** (*int*) – The number of latent variables in the model.
- **e** (*int*) – The number of environments in the model.
- **A** (*numpy.ndarray*) – The *p x p* adjacency matrix of the DAG underlying the model, where *A[i,j] != 0 implies i -> j*.
- **B** (*numpy.ndarray*) – The *p x p* connectivity (edge weights) matrix. Follows the sparsity pattern of A.
- **gamma** (*numpy.ndarray*) – The *l x p* matrix of latent effects, i.e. connectivity matrix from latent to observed variables, where *gamma[i,j] != 0* implies *i -> j*.
- **omegas** (*numpy.ndarray*) – The *e x p* matrix containing the variances of the observed variables' noise terms.
- **psis** (*numpy.ndarray*) – The *e x l* array with the variances of the latent variables for each environment.

__init__(*A*, *B*, *gamma*, *omegas*, *psis*)

Create a new instance of a model.

Parameters

- **A** (*numpy.ndarray*) – The *p x p* adjacency matrix of the given DAG, where *A[i,j] != 0 implies i -> j*.
- **B** (*numpy.ndarray*) – The *p x p* connectivity (weight) matrix.
- **gamma** (*numpy.ndarray*) – The *l x p* matrix of latent effects, i.e. connectivity matrix from latent to observed variables, where *gamma[i,j] != 0* implies *i -> j*.
- **omegas** (*numpy.ndarray*) – A *e x p* matrix containing containing the variances of the observed variables' noise terms.
- **psis** (*numpy.ndarray*) – A *e x l* array with the variances of the latent variables for each environment.

Return type NoneType

Raises **ValueError :** – If A is not a DAG adjacency or if B does not respect the sparsity pattern in A; if the dimensions of the different parameters are not compatible.

Examples

Creating an instance of a model with 3 observed variables, 2 latents and 5 environments.

```
>>> rng = np.random.default_rng(42)
>>> A = np.array([[0,0,1], [0,0,1], [0,0,0]])
>>> B = np.array([[0,0,0.5], [0,0,3], [0,0,0]])
>>> gamma = rng.uniform(size=(2,3))
>>> omegas = rng.uniform(size=(5,3))
>>> psis = rng.uniform(size=(5,2))
>>> model = Model(A, B, gamma, omegas, psis)
```

```
>>> model.A
array([[0, 0, 1],
       [0, 0, 1],
       [0, 0, 0]])
```

```
>>> model.B
array([[0. , 0. , 0.5],
       [0. , 0. , 3. ],
       [0. , 0. , 0. ]])
```

```
>>> model.gamma
array([[0.77395605, 0.43887844, 0.85859792],
       [0.69736803, 0.09417735, 0.97562235]])
```

```
>>> model.omegas
array([[0.7611397 , 0.78606431, 0.12811363],
       [0.45038594, 0.37079802, 0.92676499],
       [0.64386512, 0.82276161, 0.4434142 ],
       [0.22723872, 0.55458479, 0.06381726],
       [0.82763117, 0.6316644 , 0.75808774]])
```

```
>>> model.psis
array([[0.35452597, 0.97069802],
       [0.89312112, 0.7783835 ],
       [0.19463871, 0.466721  ],
       [0.04380377, 0.15428949],
       [0.68304895, 0.74476216]])
```

```
>>> model.p
3
```

```
>>> model.e
5
```

```
>>> model.l
2
```

```
>>> model.I_B
array([[ 1. ,  0. ,  0. ],
```

(continues on next page)

(continued from previous page)

```
       [ 0. ,  1. ,  0. ],
       [-0.5, -3. ,  1. ]])
```

When A is not a DAG: >>> bad_A = np.array([[0,0,1], [0,0,1], [1,0,0]]) >>> Model(bad_A, B, gamma, omegas, psis) Traceback (most recent call last): ... ValueError: A does not correspond to a DAG.

When B does not match the sparsity pattern: >>> Model(A, B.T, gamma, omegas, psis) Traceback (most recent call last): ... ValueError: B does not respect sparsity pattern in A.

When the dimensions of the parameters are incompatible:

```
>>> bad_B = np.array([[0,0.5], [0,3]])
>>> Model(A, bad_B, gamma, omegas, psis)
Traceback (most recent call last):
...
ValueError: A and B have different dimensions.
```

```
>>> bad_gamma = rng.uniform(size=(2,4))
>>> Model(A, B, bad_gamma, omegas, psis)
Traceback (most recent call last):
...
ValueError: The sizes of A and gamma are not compatible.
```

```
>>> bad_omegas = rng.uniform(size=(5,2))
>>> Model(A, B, gamma, bad_omegas, psis)
Traceback (most recent call last):
...
ValueError: The sizes of A and omegas are not compatible.
```

```
>>> bad_psis = rng.uniform(size=(4,2))
>>> Model(A, B, gamma, omegas, bad_psis)
Traceback (most recent call last):
...
ValueError: The sizes of omegas and psis are not compatible.
```

```
>>> bad_psis = rng.uniform(size=(5,3))
>>> Model(A, B, gamma, omegas, bad_psis)
Traceback (most recent call last):
...
ValueError: The sizes of gamma and psis are not compatible.
```

copy()

Returns a copy of the current model.

Returns **copy** – A copy of this object. All contained arrays are copied using ndarray.copy().

Return type *Model*()

Example

```
>>> model.psis
array([[0.35452597, 0.97069802],
       [0.89312112, 0.7783835 ],
       [0.19463871, 0.466721  ],
       [0.04380377, 0.15428949],
       [0.68304895, 0.74476216]])
>>> copy = model.copy()
>>> copy.psis
array([[0.35452597, 0.97069802],
       [0.89312112, 0.7783835 ],
       [0.19463871, 0.466721  ],
       [0.04380377, 0.15428949],
       [0.68304895, 0.74476216]])
```

covariances()

The covariance matrices of the observed variables, as entailed by the model in each environment.

Returns **covariances** – The covariance matrices.

Return type numpy.ndarray

Example

```
>>> model.covariances()
array([[[ 1.44557555,  0.18417459,  2.17133182],
        [ 0.18417459,  0.86296055,  2.90375069],
        [ 2.17133182,  2.90375069, 12.22668828]],
...
```

intervention_strength()

Assumption deviation metric: we described how approximate knowledge of the latent variables is enough for identifiability as long as the interventions on the observed variables are strong. Thus, as a second indicator, we measure the strength of the interventions. See section 3.5 of the paper for more information.

Returns **metric** – An array of floats with as many entries as variables (p) in the model, indicating the strength of the interventions over each observed variable.

Return type numpy.ndarray

Examples

```
>>> model.intervention_strength()
array([0.0578593 , 0.03265554, 0.12387794])
```

inv_noise_term_covariances()

Compute the inverse noise term covariance matrix, noted as M, for each environment.

Returns **Ms** – A $e \; x \; p \; x \; p$ array containing the inverse noise-term covariance matrices, one per environment.

Return type numpy.ndarray

Example

```
>>> model.inv_noise_term_covariances()
array([[[ 1.20040982, -0.04683013, -0.81098407],
        [-0.04683013,  1.21369509, -0.1739194 ],
        [-0.81098407, -0.1739194 ,  1.34413286]],
...
```

noise_term_covariances()

Compute the noise-term covariance matrix for each environment.

Returns **noise_term_covariances** – A *e x p x p* array containing the inverse noise-term covariance matrices, one per environment.

Return type numpy.ndarray

Example

```
>>> model.noise_term_covariances()
array([[[1.44557555, 0.18417459, 0.89602027],
        [0.18417459, 0.86296055, 0.22278173],
        [0.89602027, 0.22278173, 1.31341498]],
...
```

sample(*n_obs*, *compute_covs=False*, *random_state=42*)

Generate a multi-environment sample from the model.

Parameters

- **n_obs** (*int or array-like of ints*) – The number of observations to generate from each environment. If a single number is passed, generate this number of observations for all environments.
- **compute_covs** (*bool, default=False*) – If additionally the sample_covariances for the generated samples should be computed.
- **random_state** (*NoneType or int, default=42*) – To set the random state for reproducibility. If *None*, subsequent calls will yield different samples.

Returns

- **X** (*list of numpy.ndarray*) – A list containing the sample from each environment.
- **sample_covariances** (*numpy.ndarray*) – A 3-dimensional array containing the estimated sample covariances of the observed variables for each environment. Returned only if *compute_covs=True*.
- **n_obs** (*numpy.nadarray of ints*) – The number of observations available from each environment (i.e. the sample size). Returned only if *compute_covs=True*.

Raises

- **ValueError :** – If the values passed for n_obs are not positive, the length of n_obs does not match the number of environments, or *sample_covariances=True* but we are sampling a single observation from any of the environments (i.e. covariance matrix cannot be computed).
- **TypeError :** – If n_obs is not a list of integers.

Examples

Generating a random sample:

```
>>> model.sample(10)
[array([[-1.30178026, -0.03529043, -0.90999532],
...
```

Additionally computing the sample covariances:

```
>>> X, covariances, n_obs = model.sample(10, compute_covs=True)
>>> n_obs
array([10, 10, 10, 10, 10])
>>> covariances
array([[[ 1.70009515,  0.05345342,  2.35152191],
        [ 0.05345342,  0.21506513,  0.67053129],
        [ 2.35152191,  0.67053129,  5.20810612]],
...
```

We cannot compute the sample covariances when the sample contains a single observation:

```
>>> model.sample(1)
[array([[-1.30178026, -0.03529043, -0.90999532]]),...
>>> model.sample(1, compute_covs=True)
Traceback (most recent call last):
...
ValueError: Cannot compute sample covariances for a single observation.
>>> model.sample([1,2,3,4,5], compute_covs=True)
Traceback (most recent call last):
...
ValueError: Cannot compute sample covariances for a single observation.
```

Specifying a different number of observations per environment:

```
>>> model.sample([2,3,4,5,6])
[array([[-1.30178026, -0.03529043, -0.90999532],
...
```

Examples of failure (Value Errors)

```
>>> model.sample([1,2])
Traceback (most recent call last):
...
ValueError: n_obs has the wrong length.
>>> model.sample([-1,2,3,4,5])
Traceback (most recent call last):
...
ValueError: n_obs should be a positive integer or list of positive integers.
>>> model.sample([0,2,3,4,5])
Traceback (most recent call last):
...
ValueError: n_obs should be a positive integer or list of positive integers.
>>> model.sample(0)
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
ValueError: n_obs should be a positive integer or list of positive integers.
```

Examples of failure (Type Errors):

```
>>> model.sample([1.0,2,3,4,5])
Traceback (most recent call last):
...
TypeError: n_obs should be a positive integer or list of positive integers.
>>> model.sample("a")
Traceback (most recent call last):
...
TypeError: n_obs should be a positive integer or list of positive integers.
```

scaled_latent_incoherence()

Assumption deviation metric: motivated by the incoherence (denseness) assumption of the latent effects, the measure computes the incoherence of the latent effects estimated by the model. See section 3.5 of the paper for more information.

We output the latent incoherence scaled by the max. degree of the moral graph.

Returns **metric** – The latent incoherence metric of the model.

Return type float

Examples

```
>>> model.scaled_latent_incoherence()
1.9462534859894438
```

score(*sample_covariances*, *n_obs*)

Compute the score of the model for the given sample covariances and number of observations from each environment.

Parameters

- **sample_covariances** (*numpy.ndarray*) – A 3-dimensional array containing the estimated sample covariances of the observed variables for each environment.
- **n_obs** (*list of ints*) – The number of observations available from each environment (i.e. the sample size).

Returns **score** – The computed score.

Return type float

Examples

```
>>> model.score(sample_covariances, n_obs)
1.1070517672870...
```

4.3 utlvce.score

The *utlvce.score* module contains the implementation of the alternating optimization procedure described in the paper, which is used to fit a UT-LVCE model to the data and compute its likelihood.

The procedure is accessed through the *utlvce.score.Score* class, which contains a caching mechanism (see class utlvce.score._Cache) to avoid re-running the procedure for the same DAG / intervention targets. For more details on the implementation please refer to section 4.1 of the paper and to the source-code.

class utlvce.score.**Score**(*data*, *num_latent*, *psi_max*, *psi_fixed*, *max_iter*, *threshold_dist_B*, *threshold_fluctuation*, *max_fluctuations*, *threshold_score*, *learning_rate*, *B_solver='grad'*, *cache=True*)

Contains the implementation of the alternating optimization procedure described in the paper; which fits a UT-LVC model given a DAG adjacency and intervention targets *I*.

Parameters

- **sample_covariances** (*numpy.ndarray*) – A 3-dimensional array containing the estimated sample covariances of the observed variables for each environment.
- **n_obs** (*list of ints*) – The number of observations available from each environment (i.e. the sample size).
- **num_latent** (*int*) – The assumed number of hidden variables.
- **psi_max** (*float, default = None*) – The maximum allowed change in variance between environments for the hidden variables. If None, psis are unconstrained.
- **psi_fixed** (*bool*) – If *True*, impose the additional constraint that the hidden variables have all the same variance.
- **max_iter** (*int*) – The maximum number of iterations allowed for the alternating minimization procedure.
- **threshold_dist_B** (*float*) – If the change in B between successive iterations of the alternating optimization procedure is lower than this threshold, stop the procedure and return the estimate.
- **threshold_fluctuation** (*float*) – For the alternating optimization routine, if the score worsens by more than this value, consider the iteration a fluctuation.
- **max_fluctuations** (*int*) – For the alternating optimization routine, the maximum number of fluctuations(see above) allowed before stopping the subroutine.
- **threshold_score** (*float*) – For the gradient descent subroutines, if change in score between successive iterations is below this threshold, stop.
- **learning_rate** (*float*) – The initial learning rate(factor by which gradient is multiplied) for the gradient descent subroutines.
- **cache** (*_Cache or None*) – The cache used to store results of calls to score_dag().

__init__(*data*, *num_latent*, *psi_max*, *psi_fixed*, *max_iter*, *threshold_dist_B*, *threshold_fluctuation*, *max_fluctuations*, *threshold_score*, *learning_rate*, *B_solver='grad'*, *cache=True*)

Create a new instance of a Score object.

Parameters

- **data** (*list of numpy.ndarray or tuple or numpy.ndarray*) –

  Can be either:

  1) A list with the samples from each environment, where each sample is an array with columns corresponding to variables and rows to observations.

  2) A tuple containing the precomputed sample covariances and number of observations from each environment.

- **num_latent** (*int*) – The assumed number of hidden variables.
- **psi_max** (*float, default = None*) – The maximum allowed change in variance between environments for the hidden variables. If None, psis are unconstrained.
- **psi_fixed** (*bool*) – If *True*, impose the additional constraint that the hidden variables have all the same variance.
- **max_iter** (*int*) – The maximum number of iterations allowed for the alternating minimization procedure.
- **threshold_dist_B** (*float*) – If the change in B between successive iterations of the alternating optimization procedure is lower than this threshold, stop the procedure and return the estimate.
- **threshold_fluctuation** (*float*) – For the alternating optimization routine, if the score worsens by more than this value, consider the iteration a fluctuation.
- **max_fluctuations** (*int*) – For the alternating optimization routine, the maximum number of fluctuations(see above) allowed before stopping the subroutine.
- **threshold_score** (*float*) – For the gradient descent subroutines, if change in score between successive iterations is below this threshold, stop.
- **learning_rate** (*float*) – The initial learning rate(factor by which gradient is multiplied) for the gradient descent subroutines.
- **B_solver** (*{'grad', 'adaptive', 'cvx'}, default='grad'*) – Sets the solver for the connectivity matrix B, where the options are (ordered by decreasing speed and increasing stability) *grad*, *adaptive* and *cvx*.
- **cache** (*bool, default=True*) – If results from calling the *score_dag* function should be cached.

Raises

- **ValueError :** – If the given data is not valid, i.e. (different number of variables per sample, or one sample with a single observation). Also if the given *B_solver* is not valid.
- **TypeError :** – If the given data is of invalid type.

Examples

Initializing the *utlvce.score.Score* using the "raw" data:

```
>>> score_params = {'psi_max': None,
...                 'psi_fixed': False,
...                 'max_iter': 1000,
...                 'threshold_dist_B': 1e-4,
...                 'threshold_fluctuation': 1,
...                 'max_fluctuations': 10,
...                 'threshold_score': 1e-5,
...                 'learning_rate': 1,
...                 'cache': True}
>>> data = list(rng.uniform(size=(5,1000,20)))
>>> Score(data, num_latent=2, **score_params)
<__main__.Score object at 0x...>
```

Or passing pre-computed sample covariances:

```
>>> n_obs = np.array([len(sample) for sample in data])
>>> sample_covariances = np.array([np.cov(sample, rowvar=False) for sample in␣
↪data])
>>> Score((sample_covariances, n_obs), num_latent=2, **score_params)
<__main__.Score object at 0x...>
```

Errors are raised when not all samples have the same number of variables:

```
>>> bad_data = data.copy()
>>> bad_data[0] = bad_data[0][:,:-1]
>>> Score(bad_data, num_latent=2, **score_params)
Traceback (most recent call last):
...
ValueError: All samples must have the same number of variables.
```

```
>>> n_obs = np.array([len(sample) for sample in bad_data])
>>> sample_covariances = np.array([np.cov(sample, rowvar=False) for sample in␣
↪bad_data])
>>> Score((sample_covariances, n_obs), num_latent=2, **score_params)
Traceback (most recent call last):
...
ValueError: All samples must have the same number of variables.
```

Or when one sample has a single observation:

```
>>> bad_data = data.copy()
>>> bad_data[0] = bad_data[0][[0], :]
>>> Score(bad_data, num_latent=2, **score_params)
Traceback (most recent call last):
...
ValueError: Each sample must contain at least two observations to estimate the␣
↪covariance matrix.
```

Error when wrongly selecting the solver for B: >>> Score(data, num_latent=2, B_solver='test', **score_params) Traceback (most recent call last): ... ValueError: Unrecognized value "test" for parameter B_solver.

score_dag(*A*, *I*, *init_model=None*, *verbose=0*)

Score the given DAG while allowing interventions on certain variables.

Parameters

- **A** (*numpy.ndarray*) – The adjacency of the given DAG, where *A[i, j] != 0 implies i -> j*.
- **I** (*set*) – The observed variables for which interventions are allowed, i.e. for which the noise term distribution is allowed to change between environments.
- **init_model** (*utlvce.Model, default = None*) – To set a set of parameters, encoded by an instance of utlvce.model.Model as the starting point of the procedure.
- **verbose** (*int, default = 0*) – If debug and execution traces should be printed. *0* corresponds to no traces, higher values correspond to higher verbosity.

Raises **ValueError:** – If the given adjacency does not correspond to a DAG.

Returns

- **model** (*utlvce.Model*) – The estimated model with all nuisance parameters. *model.B* returns the connectivity(weight) matrix found to maximize the score.
- **score** (*float*) – The score attained by the given DAG adjacency.

4.4 utlvce.generators

The *utlvce.generators* module contains functions to generate random UT-LVCE models from given or random DAG adjacencies.

utlvce.generators.**chain_graph_model**(*p*, *I*, *num_latent*, *e*, *var_lo*, *var_hi*, *int_var_lo*, *int_var_hi*, *psi_lo*, *psi_hi*, *int_psi_lo*, *int_psi_hi*, *B_lo*, *B_hi*, *sparse_latents=False*, *obs=True*, *random_state=42*, *verbose=0*)

Generate a random model from a chain graph with *p* nodes.

Parameters

- **p** (*int*) – The number of observed variables in the model.
- **I** (*set*) – The set of intervention targets.
- **num_latent** (*int*) – The number of latent variables in the model.
- **e** (*int*) – The number of environments.
- **var_lo** (*float*) – The lower bound for the variances of the noise terms of the observed variables.
- **var_hi** (*float*) – The upper bound for the variances of the noise terms of the observed variables.
- **int_var_lo** (*float*) – The lower bound for the intervention variances on the observed variables.
- **int_var_hi** (*float*) – The upper bound for the intervention variances on the observed variables.
- **psi_lo** (*float*) – The lower bound for the variances of the latent variables.
- **psi_hi** (*float*) – The upper bound for the variances of the latent variables.

- **int_psi_lo** (*float*) – The lower bound for the intervention variances on the latent variables.
- **int_psi_hi** (*float*) – The upper bound for the intervention variances on the latent variables.
- **B_lo** (*float*) – The lower bound for the edge weights between observed variables.
- **B_hi** (*float*) – The upper bound for the edge weights between observed variables.
- **sparse_latents** (*bool, default=False*) – If the gamma matrix of latent effects should be sparse (see source).
- **obs** (*bool, default=True*) – Whether the first environment should be "observational", i.e. that the variances of the noise terms and latents are lower (variable-wise) than the other environments. With *obs=True*, the variances for first environment are sampled from *[var_lo, var_hi]* and, from *[var_lo + int_var_lo, var_hi + int_var_hi]* for the remaining environments; the same holds for the sampling of *psi*. If *obs=False*, the latter interval is used for all environments. Note that is not a necessary assumption for the UT-LVCE estimator, but makes the actual intervention strength less sensitive to the random sampling of parameters.
- **random_state** (*int, default=42*) – To set the random state for reproducibility. Successive calls with the same random state will return the same model.
- **verbose** (*int, default = 0*) – If debug and execution traces should be printed. *0* corresponds to no traces, higher values correspond to higher verbosity.

Returns **model** – An instance of the model with the sampled parameters.

Return type *utlvce.model.Model*

Raises **ValueError :** – If the intervention targets are not a subset of the variable indices, i.e. *[0,...,p-1]*.

Examples

```
>>> chain_graph_model(20,{2},2,5,0.5,0.6,3,6,0.2,0.4,1,5,0.7,0.8,False,True,42,0)
<utlvce.model.Model object at 0x...>
```

utlvce.generators.**intervention_targets**(*p*, *num_targets*, *random_state=42*)

Sample a set of intervention targets.

Parameters

- **p** (*int*) – The number of variables, i.e. targets will be sampled from *[0,p-1]*.
- **num_targets** (*int or tuple*) – Specifies the number of targets. If a two-element tuple, the number of targets is sampled uniformly at random from *[size[0], size[1]]*
- **random_state** (*int*) – To set the random state for reproducibility.

Returns **targets** – A set with the indices of the intervention targets.

Return type set

Raises **ValueError :** – If the given number of targets is invalid.

Examples

```
>>> intervention_targets(20, 3)
{1, 13, 14}
```

```
>>> intervention_targets(20, (1,10), random_state=1)
{0, 2, 8, 12, 17}
```

```
>>> intervention_targets(20, (1,10), random_state=2)
{1, 3, 4, 6, 8, 13, 18, 19}
```

```
>>> intervention_targets(10, 10)
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
>>> intervention_targets(10, 0)
set()
```

Requesting an inappropriate (>p) number of targets yields a *ValueError*:

```
>>> intervention_targets(10, 11)
Traceback (most recent call last):
...
ValueError: Invalid number of targets.
```

```
>>> intervention_targets(10, (0,11))
Traceback (most recent call last):
...
ValueError: Invalid number of targets.
```

utlvce.generators.**random_graph_model**(*p*, *k*, *I*, *num_latent*, *e*, *var_lo*, *var_hi*, *int_var_lo*, *int_var_hi*, *psi_lo*, *psi_hi*, *int_psi_lo*, *int_psi_hi*, *B_lo*, *B_hi*, *sparse_latents=False*, *obs=True*, *random_state=42*, *verbose=0*)

Generate a random model from a random Erdős–Rényi graph with *p* nodes and average degree *k*.

Parameters

- **p** (*int*) – The number of observed variables in the model.
- **k** (*float*) – The average degree of the underlying Erdős–Rényi graph.
- **I** (*set*) – The set of intervention targets.
- **num_latent** (*int*) – The number of latent variables in the model.
- **e** (*int*) – The number of environments.
- **var_lo** (*float*) – The lower bound for the variances of the noise terms of the observed variables.
- **var_hi** (*float*) – The upper bound for the variances of the noise terms of the observed variables.
- **int_var_lo** (*float*) – The lower bound for the intervention variances on the observed variables.
- **int_var_hi** (*float*) – The upper bound for the intervention variances on the observed variables.

- **psi_lo** (*float*) – The lower bound for the variances of the latent variables.
- **psi_hi** (*float*) – The upper bound for the variances of the latent variables.
- **int_psi_lo** (*float*) – The lower bound for the intervention variances on the latent variables.
- **int_psi_hi** (*float*) – The upper bound for the intervention variances on the latent variables.
- **B_lo** (*float*) – The lower bound for the edge weights between observed variables.
- **B_hi** (*float*) – The upper bound for the edge weights between observed variables.
- **sparse_latents** (*bool, default=False*) – If the gamma matrix of latent effects should be sparse (see source).
- **obs** (*bool, default=True*) – Whether the first environment should be "observational", i.e. that the variances of the noise terms and latents are lower (variable-wise) than the other environments. With *obs=True*, the variances for first environment are sampled from *[var_lo, var_hi]* and, from *[var_lo + int_var_lo, var_hi + int_var_hi]* for the remaining environments; the same holds for the sampling of *psi*. If *obs=False*, the latter interval is used for all environments. Note that is not a necessary assumption for the UT-LVCE estimator, but makes the actual intervention strength less sensitive to the random sampling of parameters.
- **random_state** (*int, default=42*) – To set the random state for reproducibility. Successive calls with the same random state will return the same model.
- **verbose** (*int, default = 0*) – If debug and execution traces should be printed. *0* corresponds to no traces, higher values correspond to higher verbosity.

Returns **model** – An instance of the model with the sampled parameters.

Return type *utlvce.model.Model*

Raises **ValueError :** – If the intervention targets are not a subset of the variable indices, i.e. *[0,...,p-1]*.

Examples

```
>>> random_graph_model(20,2.1,{2},2,5,0.5,0.6,3,6,0.2,0.4,1,5,0.7,0.8,False,True,42,
↪0)
<utlvce.model.Model object at 0x...>
```

utlvce.generators.**sample_parameters**(*A*, *I*, *num_latent*, *e*, *var_lo*, *var_hi*, *int_var_lo*, *int_var_hi*, *psi_lo*, *psi_hi*, *int_psi_lo*, *int_psi_hi*, *B_lo*, *B_hi*, *sparse_latents=False*, *obs=True*, *random_state=42*, *verbose=0*)

Generate a random model given an adjacency matrix *A* and intervention targets *I*.

Parameters

- **A** (*numpy.ndarray*) – The adjacency matrix of the DAG underlying the model, where *A[i,j] != 0* implies *i -> j*.
- **I** (*set*) – The set of intervention targets.
- **num_latent** (*int*) – The number of latent variables in the model.
- **e** (*int*) – The number of environments.

- **var_lo** (*float*) – The lower bound for the variances of the noise terms of the observed variables.
- **var_hi** (*float*) – The upper bound for the variances of the noise terms of the observed variables.
- **int_var_lo** (*float*) – The lower bound for the intervention variances on the observed variables.
- **int_var_hi** (*float*) – The upper bound for the intervention variances on the observed variables.
- **psi_lo** (*float*) – The lower bound for the variances of the latent variables.
- **psi_hi** (*float*) – The upper bound for the variances of the latent variables.
- **int_psi_lo** (*float*) – The lower bound for the intervention variances on the latent variables.
- **int_psi_hi** (*float*) – The upper bound for the intervention variances on the latent variables.
- **B_lo** (*float*) – The lower bound for the edge weights between observed variables.
- **B_hi** (*float*) – The upper bound for the edge weights between observed variables.
- **sparse_latents** (*bool, default=False*) – If the gamma matrix of latent effects should be sparse (see source).
- **obs** (*bool, default=True*) – Whether the first environment should be "observational", i.e. that the variances of the noise terms and latents are lower (variable-wise) than the other environments. With *obs=True*, the variances for first environment are sampled from *[var_lo, var_hi]* and, from *[var_lo + int_var_lo, var_hi + int_var_hi]* for the remaining environments; the same holds for the sampling of *psi*. If *obs=False*, the latter interval is used for all environments. Note that is not a necessary assumption for the UT-LVCE estimator, but makes the actual intervention strength less sensitive to the random sampling of parameters.
- **random_state** (*int, default=42*) – To set the random state for reproducibility. Successive calls with the same random state will return the same model.
- **verbose** (*int, default = 0*) – If debug and execution traces should be printed. *0* corresponds to no traces, higher values correspond to higher verbosity.

Returns **model** – An instance of the model with the sampled parameters.

Return type *utlvce.model.Model*

Raises **ValueError :** – If the given adjacency is not a DAG or the intervention targets are not a subset of the variable indices, i.e. *[0,...,p-1]*.

Examples

```
>>> A = np.array([[0, 0, 1], [0, 0, 1], [0, 0, 0]])
>>> sample_parameters(A,{2},2,5,0.5,0.6,3,6,0.2,0.4,1,5,0.7,0.8,False,True,42,0)
<utlvce.model.Model object at 0x...>
```

Requesting an inappropriate (>p) number of targets yields a *ValueError*:

```
>>> sample_parameters(A,{3},2,5,0.5,0.6,3,6,0.2,0.4,1,5,0.7,0.8,False,True,42,0)
Traceback (most recent call last):
...
ValueError: The intervention targets must be a subset of [0,...,p-1].
```

A *ValueError* is raised if the given adjacency does not correspond to a DAG (e.g. it contains cycles):

```
>>> A = np.array([[0, 0, 1], [0, 0, 1], [1, 0, 0]])
>>> sample_parameters(A,{2},2,5,0.5,0.6,3,6,0.2,0.4,1,5,0.7,0.8,False,True,42,0)
Traceback (most recent call last):
...
ValueError: The given adjacency does not correspond to a DAG.
```

PYTHON MODULE INDEX

u

utlvce.generators, 23
utlvce.score, 20

INDEX

Symbols

__init__() (*utlvce.Model method*), 13
__init__() (*utlvce.score.Score method*), 20

C

chain_graph_model() (*in module utlvce.generators*), 23
copy() (*utlvce.Model method*), 15
covariances() (*utlvce.Model method*), 16

E

equivalence_class() (*in module utlvce*), 10
equivalence_class_w_ges() (*in module utlvce*), 11

I

intervention_strength() (*utlvce.Model method*), 16
intervention_targets() (*in module utlvce.generators*), 24
inv_noise_term_covariances() (*utlvce.Model method*), 16

M

Model (*class in utlvce*), 13
module
    utlvce.generators, 23
    utlvce.score, 20

N

noise_term_covariances() (*utlvce.Model method*), 17

R

random_graph_model() (*in module utlvce.generators*), 25

S

sample() (*utlvce.Model method*), 17
sample_parameters() (*in module utlvce.generators*), 26
scaled_latent_incoherence() (*utlvce.Model method*), 19
Score (*class in utlvce.score*), 20
score() (*utlvce.Model method*), 19
score_dag() (*utlvce.score.Score method*), 23

U

utlvce.generators
    module, 23
utlvce.score
    module, 20