PyDTMC Documentation

Release 8.7.0

['Tommaso Belluzzo']

Mar 18, 2024

TABLE OF CONTENTS

CHAPTER

ONE

MARKOV CHAIN

1.1 Properties

class pydtmc.**MarkovChain**(*p*, *states=None*)

Defines a Markov chain with the given transition matrix.

Parameters

- **p** (Union[ndarray, spmatrix]) – the transition matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises

ValidationError – if any input argument is not compliant.

absorbing_states: List[List[str]]

A property representing the absorbing states of the Markov chain.

accessibility_matrix: ndarray

A property representing the accessibility matrix of the Markov chain.

adjacency_matrix: ndarray

A property representing the adjacency matrix of the Markov chain.

communicating_classes: List[List[str]]

A property representing the communicating classes of the Markov chain.

communication_matrix: ndarray

A property representing the communication matrix of the Markov chain.

cyclic_classes: List[List[str]]

A property representing the cyclic classes of the Markov chain.

cyclic_states: List[List[str]]

A property representing the cyclic states of the Markov chain.

density: float

A property representing the density of the transition matrix of the Markov chain.

determinant: float

A property representing the determinant of the transition matrix of the Markov chain.

entropy_rate: *float | None*

A property representing the entropy rate of the Markov chain.
If the Markov chain has multiple stationary distributions, then `None` is returned.

entropy_rate_normalized: *float | None*

A property representing the entropy rate, normalized between 0 and 1, of the Markov chain.
If the Markov chain has multiple stationary distributions, then `None` is returned.

fundamental_matrix: *ndarray | None*

A property representing the fundamental matrix of the Markov chain.
If the Markov chain is not **absorbing** or has no transient states, then `None` is returned.

implied_timescales: *ndarray | None*

A property representing the implied timescales of the Markov chain.
If the Markov chain is not **ergodic**, then `None` is returned.

incidence_matrix: *ndarray*

A property representing the incidence matrix of the Markov chain.

is_absorbing: *bool*

A property indicating whether the Markov chain is absorbing.

is_aperiodic: *bool*

A property indicating whether the Markov chain is aperiodic.

is_canonical: *bool*

A property indicating whether the Markov chain has a canonical form.

is_doubly_stochastic: *bool*

A property indicating whether the Markov chain is doubly stochastic.

is_ergodic: *bool*

A property indicating whether the Markov chain is ergodic.

is_irreducible: *bool*

A property indicating whether the Markov chain is irreducible.

is_regular: *bool*

A property indicating whether the Markov chain is regular.

is_reversible: *bool*

A property indicating whether the Markov chain is reversible.

is_stochastically_monotone: *bool*

A property indicating whether the Markov chain is stochastically monotone.

is_symmetric: *bool*

A property indicating whether the Markov chain is symmetric.

kemeny_constant: float | None

A property representing the Kemeny's constant of the fundamental matrix of the Markov chain.
If the Markov chain is not **absorbing** or has no transient states, then None is returned.

lumping_partitions: List[List[List[int]] | List[List[str]]]

A property representing all the partitions of the Markov chain that satisfy the ordinary lumpability criterion.

mixing_rate: float | None

A property representing the mixing rate of the Markov chain.
If the Markov chain is not **ergodic** or the **SLEM** (second largest eigenvalue modulus) cannot be computed, then None is returned.

n: int

A property representing the size of the Markov chain state space.

p: ndarray

A property representing the transition matrix of the Markov chain.

period: int

A property representing the period of the Markov chain.

periods: List[int]

A property representing the period of each communicating class defined by the Markov chain.

pi: List[ndarray]

A property representing the stationary distributions of the Markov chain.
Aliases: stationary_distributions, steady_states

rank: int

A property representing the rank of the transition matrix of the Markov chain.

recurrent_classes: List[List[str]]

A property representing the recurrent classes defined by the Markov chain.

recurrent_states: List[List[str]]

A property representing the recurrent states of the Markov chain.

relaxation_rate: float | None

A property representing the relaxation rate of the Markov chain.
If the Markov chain is not **ergodic** or the **SLEM** (second largest eigenvalue modulus) cannot be computed, then None is returned.

size: int

A property representing the size of the Markov chain.

spectral_gap: float | None

A property representing the spectral gap of the Markov chain.
If the Markov chain is not **ergodic** or the **SLEM** (second largest eigenvalue modulus) cannot be computed, then None is returned.

states: List[str]

A property representing the states of the Markov chain.

topological_entropy: float

A property representing the topological entropy of the Markov chain.

transient_classes: List[List[str]]

A property representing the transient classes defined by the Markov chain.

transient_states: List[List[str]]

A property representing the transient states of the Markov chain.

1.2 Instance Methods

class pydtmc.**MarkovChain**(*p*, *states=None*)

Defines a Markov chain with the given transition matrix.

Parameters

- **p** (Union[ndarray, spmatrix]) – the transition matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises

ValidationError – if any input argument is not compliant.

absorption_probabilities()

The method computes the absorption probabilities of the Markov chain.

Notes:

- If the Markov chain has no transient states, then None is returned.

Return Type

Optional[ndarray]

aggregate(*s*, *method='adaptive'*)

The method attempts to reduce the state space of the Markov chain to the given number of states through a Kullback-Leibler divergence minimization approach.

Notes:

- The spectral theory based aggregation is described in Optimal Kullback-Leibler Aggregation via Spectral Theory of Markov Chains (Deng et al., 2011).

Parameters

- **s** (int) – the number of states of the reduced Markov chain.
- **method** (str)
  - **spectral-bottom-up** for a spectral theory based aggregation, bottom-up (more suitable for reducing a large number of states).

- **spectral-top-down** for a spectral theory based aggregation, top-down (more suitable for reducing a small number of states).
- **adaptive** for automatically selecting the best aggregation method.

Raises

- *ValidationError* – if any input argument is not compliant.
- ValueError – if the Markov chain defines only two states or is not **ergodic**.

Return Type
MarkovChain

are_communicating(*state1*, *state2*)

The method verifies whether the given states of the Markov chain are communicating.

Parameters

- **state1** (Union[int, str]) – the first state.
- **state2** (Union[int, str]) – the second state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
bool

closest_reversible(*initial_distribution=None*, *weighted=False*)

The method computes the closest reversible of the Markov chain.

Notes:

- The algorithm is described in Computing the Nearest Reversible Markov chain (Nielsen & Weber, 2015).

Parameters

- **initial_distribution** (Optional[Union [ndarray, spmatrix]]) – the distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).
- **weighted** (bool) – a boolean indicating whether to use the weighted Frobenius norm.

Raises

- *ValidationError* – if any input argument is not compliant.
- ValueError – if the closest reversible could not be computed.

Return Type
MarkovChain

committor_probabilities(*committor_type*, *states1*, *states2*)

The method computes the committor probabilities between the given subsets of the state space defined by the Markov chain.

Notes:

- If the Markov chain is not **ergodic**, then None is returned.

- The method can be accessed through the following aliases: **crp**.

Parameters

- **committor_type** (str)
  - **backward** for the backward committor;
  - **forward** for the forward committor.
- **states1** (Union[int, str, List[int], List[str]]) – the first subset of the state space.
- **states2** (Union[int, str, List[int], List[str]]) – the second subset of the state space.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Optional[ndarray]

conditional_probabilities(*state*)

The method computes the probabilities, for all the states of the Markov chain, conditioned on the process being at the given state.

Notes:

- The method can be accessed through the following aliases: **conditional_distribution**, **cd**, **cp**.

Parameters
state (Union[int, str]) – the current state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
ndarray

expected_rewards(*steps*, *rewards*)

The method computes the expected rewards of the Markov chain after **N** steps, given the reward value of each state.

Notes:

- The method can be accessed through the following aliases: **er**.

Parameters

- **steps** (int) – the number of steps.
- **rewards** (Union[ndarray, spmatrix]) – the reward values.

Raises
ValidationError – if any input argument is not compliant.

Return Type
ndarray

expected_transitions(*steps*, *initial_distribution=None*)

The method computes the expected number of transitions performed by the Markov chain after *N* steps, given the initial distribution of the states.

Notes:

- The method can be accessed through the following aliases: **et**.

Parameters

- **steps** (int) – the number of steps.
- **initial_distribution** (Optional[Union [ndarray, spmatrix]]) – the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).

Raises

ValidationError – if any input argument is not compliant.

Return Type

ndarray

first_passage_probabilities(*steps*, *initial_state*, *first_passage_states=None*)

The method computes the first passage probabilities of the Markov chain after *N* steps, given the initial state and, optionally, the first passage states.

Notes:

- The method can be accessed through the following aliases: **fpp**.

Parameters

- **steps** (int) – the number of steps.
- **initial_state** (Union[int, str]) – the initial state.
- **first_passage_states** (Optional[Union [int, str, List[int], List[str]]]) – the first passage states.

Raises

ValidationError – if any input argument is not compliant.

Return Type

ndarray

first_passage_reward(*steps*, *initial_state*, *first_passage_states*, *rewards*)

The method computes the first passage reward of the Markov chain after *N* steps, given the reward value of each state, the initial state and the first passage states.

Notes:

- The method can be accessed through the following aliases: **fpr**.

Parameters

- **steps** (int) – the number of steps.

- **initial_state** (Union[int, str]) – the initial state.
- **first_passage_states** (Union[int, str, List[int], List[str]]) – the first passage states.
- **rewards** (Union[ndarray, spmatrix]) – the reward values.

Raises

- *ValidationError* – if any input argument is not compliant.
- ValueError – if the Markov chain defines only two states.

Return Type
float

hitting_probabilities(*targets=None*)

The method computes the hitting probability, for the states of the Markov chain, to the given set of states.

Notes:

- The method can be accessed through the following aliases: **hp**.

Parameters
targets (Optional[Union [int, str, List[int], List[str]]]) – the target states (*if omitted, all the states are targeted*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
ndarray

hitting_times(*targets=None*)

The method computes the hitting times, for all the states of the Markov chain, to the given set of states.

Notes:

- The method can be accessed through the following aliases: **ht**.

Parameters
targets (Optional[Union [int, str, List[int], List[str]]]) – the target states (*if omitted, all the states are targeted*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
ndarray

is_absorbing_state(*state*)

The method verifies whether the given state of the Markov chain is absorbing.

Parameters
state (Union[int, str]) – the target state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
bool

`is_accessible`(*state_target*, *state_origin*)

The method verifies whether the given target state is reachable from the given origin state.

Parameters

- **state_target** (Union[int, str]) – the target state.
- **state_origin** (Union[int, str]) – the origin state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
bool

`is_cyclic_state`(*state*)

The method verifies whether the given state is cyclic.

Parameters
state (Union[int, str]) – the target state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
bool

`is_recurrent_state`(*state*)

The method verifies whether the given state is recurrent.

Parameters
state (Union[int, str]) – the target state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
bool

`is_transient_state`(*state*)

The method verifies whether the given state is transient.

Parameters
state (Union[int, str]) – the target state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
bool

`lump`(*partitions*)

The method attempts to reduce the state space of the Markov chain with respect to the given partitions following the ordinary lumpability criterion.

Parameters
partitions (Union[List[List[int]], List[List[str]]]) – the partitions of the state space.

Raises

- *ValidationError* – if any input argument is not compliant.
- ValueError – if the Markov chain defines only two states or is not lumpable with respect to the given partitions.

Return Type
MarkovChain

mean_absorption_times()

The method computes the mean absorption times of the Markov chain.

Notes:

- If the Markov chain is not **absorbing** or has no transient states, then None is returned.
- The method can be accessed through the following aliases: **mat**.

Return Type
Optional[ndarray]

mean_first_passage_times_between(*origins*, *targets*)

The method computes the mean first passage times between the given subsets of the state space.

Notes:

- If the Markov chain is not **ergodic**, then None is returned.
- The method can be accessed through the following aliases: **mfpt_between**, **mfptb**.

Parameters
- **origins** (Union[int, str, List[int], List[str]]) – the origin states.
- **targets** (Union[int, str, List[int], List[str]]) – the target states.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Optional[float]

mean_first_passage_times_to(*targets=None*)

The method computes the mean first passage times, for all the states, to the given set of states.

Notes:

- If the Markov chain is not **ergodic**, then None is returned.
- The method can be accessed through the following aliases: **mfpt_to**, **mfptt**.

Parameters
targets (Optional[Union [int, str, List[int], List[str]]]) – the target states (*if omitted, all the states are targeted*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
Optional[ndarray]

mean_number_visits()

The method computes the mean number of visits of the Markov chain.

Notes:

- The method can be accessed through the following aliases: **mnv**.

Return Type
Optional[ndarray]

mean_recurrence_times()

The method computes the mean recurrence times of the Markov chain.

Notes:

- If the Markov chain is not **ergodic**, then None is returned.
- The method can be accessed through the following aliases: **mrt**.

Return Type
Optional[ndarray]

merge_with(*other*, *gamma*)

The method returns a Markov chain whose transition matrix is defined below.

$$p_{new} = (1 - \gamma)p_{current} + \gamma p_{other}$$

Parameters

- **other** (*MarkovChain*) – the other Markov chain to be merged.
- **gamma** (float) – the merger blending factor.

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

mixing_time(*initial_distribution=None*, *jump=1*, *cutoff_type='natural'*)

The method computes the mixing time of the Markov chain, given the initial distribution of the states.

Notes:

- If the Markov chain is not **ergodic**, then None is returned.

- The method can be accessed through the following aliases: **mt**.

Parameters

- **initial_distribution** (Optional[Union [ndarray, spmatrix]]) – the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).
- **jump** (int) – the number of steps in each iteration.
- **cutoff_type** (str)
  - **natural** for the natural cutoff;
  - **traditional** for the traditional cutoff.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Optional[int]

next(*initial_state*, *output_index=False*, *seed=None*)

The method simulates a single step in a random walk.

Parameters

- **initial_state** (Union[int, str]) – the initial state.
- **output_index** (bool) – a boolean indicating whether to output the state index.
- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Union[int, str]

predict(*steps*, *initial_state*, *output_indices=False*)

The method computes the most probable sequence of states produced by a random walk of N steps, given the initial state.

Notes:

- In presence of probability ties None is returned.

Parameters

- **steps** (int) – the number of steps.
- **initial_state** (Union[int, str]) – the initial state.
- **output_indices** (bool) – a boolean indicating whether to output the state indices.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Optional[Union [List[int], List[str]]]

redistribute(*steps*, *initial_status=None*, *output_last=True*)

The method performs a redistribution of states of *N* steps.

Parameters

- **steps** (int) – the number of steps.
- **initial_status** (Optional[Union [int, str, ndarray, spmatrix]]) – the initial state or the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).
- **output_last** (bool) – a boolean indicating whether to output only the last distributions.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Union[ndarray, List[ndarray]]

sensitivity(*state*)

The method computes the sensitivity matrix of the stationary distribution with respect to a given state.

Notes:

- If the Markov chain is not **irreducible**, then None is returned.

Parameters

state (Union[int, str]) – the target state.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Optional[ndarray]

sequence_probability(*sequence*)

The method computes the probability of a given sequence of states.

Parameters

sequence (Union[List[int], List[str]]) – the observed sequence of states.

Raises

ValidationError – if any input argument is not compliant.

Return Type

float

simulate(*steps*, *initial_state=None*, *final_state=None*, *output_indices=False*, *seed=None*)

The method simulates a random walk of the given number of steps.

Parameters

- **steps** (int) – the number of steps.
- **initial_state** (Optional[Union [int, str]]) – the initial state (*if omitted, it is chosen uniformly at random*).
- **final_state** (Optional[Union [int, str]]) – the final state of the walk (*if specified, the simulation stops as soon as it is reached even if not all the steps have been performed*).
- **output_indices** (bool) – a boolean indicating whether to output the state indices.

- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Union[List[int], List[str]]

time_correlations(*sequence1*, *sequence2=None*, *time_points=1*)

The method computes the time autocorrelations of a single observed sequence of states or the time cross-correlations of two observed sequences of states.

Notes:

- If the Markov chain has multiple stationary distributions, then None is returned.
- If a single time point is provided, then a float is returned.
- The method can be accessed through the following aliases: **tc**.

Parameters

- **sequence1** (Union[List[int], List[str]]) – the first observed sequence of states.
- **sequence2** (Optional[Union [List[int], List[str]]]) – the second observed sequence of states.
- **time_points** (Union[int, List[int]]) – the time point or a list of time points at which the computation is performed.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Optional[Union [float, List[float]]]

time_relaxations(*sequence*, *initial_distribution=None*, *time_points=1*)

The method computes the time relaxations of an observed sequence of states with respect to the given initial distribution of the states.

Notes:

- If the Markov chain has multiple stationary distributions, then None is returned.
- If a single time point is provided, then a float is returned.
- The method can be accessed through the following aliases: **tr**.

Parameters

- **sequence** (Union[List[int], List[str]]) – the observed sequence of states.
- **initial_distribution** (Optional[Union [ndarray, spmatrix]]) – the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).
- **time_points** (Union[int, List[int]]) – the time point or a list of time points at which the computation is performed.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Optional[Union [float, List[float]]]

`to_bounded_chain`(*boundary_condition*)

The method returns a bounded Markov chain by adjusting the transition matrix of the original process using the specified boundary condition.

Notes:

- The method can be accessed through the following aliases: **to_bounded**.

Parameters
boundary_condition (Union[float, int, str])

- a number representing the first probability of the semi-reflecting condition;
- a string representing the boundary condition type (either absorbing or reflecting).

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

`to_canonical_form`()

The method returns the canonical form of the Markov chain.

Notes:

- The method can be accessed through the following aliases: **to_canonical**.

Return Type
MarkovChain

`to_dictionary`()

The method returns a dictionary representing the Markov chain.

Return Type
Dict[Tuple[str, str], float]

`to_file`(*file_path*)

The method writes a Markov chain to the given file.

Only **csv**, **json**, **txt** and **xml** files are supported; data format is inferred from the file extension.

Parameters
file_path (Union[str, Path]) – the location of the file in which the Markov chain must be written.

Raises

- OSError – if the file cannot be written.
- *ValidationError* – if any input argument is not compliant.

`to_graph`()

The method returns a directed graph representing the Markov chain.

Return Type
DiGraph

`to_lazy_chain`(*inertial_weights=0.5*)

The method returns a lazy Markov chain by adjusting the state inertia of the original process.

Notes:

- The method can be accessed through the following aliases: **to_lazy**.

Parameters
inertial_weights (Union[float, int, ndarray, spmatrix]) – the inertial weights to apply for the transformation.

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

`to_matrix`()

The method returns the transition matrix of the Markov chain.

Return Type
ndarray

`to_subchain`(*states*)

The method returns a subchain containing all the given states plus all the states reachable from them.

Notes:

- The method can be accessed through the following aliases: **to_sub**.

Parameters
states (Union[int, str, List[int], List[str]]) – the states to include in the subchain.

Raises
- *ValidationError* – if any input argument is not compliant.
- ValueError – if the subchain is not a valid Markov chain.

Return Type
MarkovChain

`transition_probability`(*state_target*, *state_origin*)

The method computes the probability of a given state, conditioned on the process being at a given state.

Parameters
- **state_target** (Union[int, str]) – the target state.

- **state_origin** (Union[int, str]) – the origin state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
float

1.3 Static Methods

class pydtmc.**MarkovChain**(*p*, *states=None*)

Defines a Markov chain with the given transition matrix.

Parameters

- **p** (Union[ndarray, spmatrix]) – the transition matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises
ValidationError – if any input argument is not compliant.

approximation(*size*, *approximation_type*, *alpha*, *sigma*, *rho*, *states=None*, *k=None*)

The method approximates the Markov chain associated with the discretized version of the first-order autoregressive process defined below.

$$y_t = (1 - \rho)\alpha + \rho y_{t-1} + \varepsilon_t$$
with $\varepsilon_t \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma_\varepsilon^2)$

Parameters

- **size** (int) – the size of the Markov chain.
- **approximation_type** (str)
  - **adda-cooper** for the Adda-Cooper approximation;
  - **rouwenhorst** for the Rouwenhorst approximation;
  - **tauchen** for the Tauchen approximation;
  - **tauchen-hussey** for the Tauchen-Hussey approximation.
- **alpha** (float) – the constant term α, representing the unconditional mean of the process.
- **sigma** (float) – the standard deviation of the innovation term ε.
- **rho** (float) – the autocorrelation coefficient ρ, representing the persistence of the process across periods.
- **k** (Optional[float])
  - In the Tauchen approximation, the number of standard deviations to approximate out to (*if omitted, the value is set to 3*).
  - In the Tauchen-Hussey approximation, the standard deviation used for the gaussian quadrature (*if omitted, the value is set to an optimal default*).

- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises

- *ValidationError* – if any input argument is not compliant.
- ValueError – if the gaussian quadrature fails to converge in the Tauchen-Hussey approximation.

Return Type
MarkovChain

birth_death(*p*, *q*, *states=None*)

The method generates a birth-death Markov chain of given size and from given probabilities.

Parameters

- **q** (ndarray) – the creation probabilities.
- **p** (ndarray) – the annihilation probabilities.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

dirichlet_process(*size*, *diffusion_factor*, *states=None*, *diagonal_bias_factor=None*, *shift_concentration=False*, *seed=None*)

The method generates a Markov chain of given size using a parametrized Dirichlet process.

Parameters

- **size** (int) – the size of the Markov chain.
- **diffusion_factor** (int) – the diffusion factor of the Dirichlet process.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).
- **diagonal_bias_factor** (Optional[float]) – the bias factor applied to the diagonal of the transition matrix (*if omitted, no inside-state stability is enforced*).
- **shift_concentration** (bool) – a boolean indicating whether to shift the concentration of the Dirichlet process to the rightmost states.
- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

fit_function(*quadrature_type*, *possible_states*, *f*, *quadrature_interval=None*)

The method fits a Markov chain using the given transition function and the given quadrature type for the computation of nodes and weights.

Notes:

- The transition function takes the four input arguments below and returns a numeric value:
  - **x_index** an integer value representing the index of the i-th quadrature node;
  - **x_value** a float value representing the value of the i-th quadrature node;
  - **y_index** an integer value representing the index of the j-th quadrature node;
  - **y_value** a float value representing the value of the j-th quadrature node.

Parameters

- **quadrature_type** (`str`)
  - **gauss-chebyshev** for the Gauss-Chebyshev quadrature;
  - **gauss-legendre** for the Gauss-Legendre quadrature;
  - **niederreiter** for the Niederreiter equidistributed sequence;
  - **newton-cotes** for the Newton-Cotes quadrature;
  - **simpson-rule** for the Simpson rule;
  - **trapezoid-rule** for the Trapezoid rule.
- **possible_states** (`List[str]`) – the possible states of the process.
- **f** (`Callable[[int, float, int, float], float]`) – the transition function of the process.
- **quadrature_interval** (`Optional[Tuple[Union[float, int], Union[float, int]]]`) – the quadrature interval to use for the computation of nodes and weights (*if omitted, the interval [0, 1] is used*).

Raises

- *ValidationError* – if any input argument is not compliant.
- `ValueError` – if the Gauss-Legendre quadrature fails to converge.

Return Type
MarkovChain

fit_sequence(*fitting_type*, *possible_states*, *sequence*, *fitting_param=None*)

The method fits a Markov chain from an observed sequence of states using the specified fitting approach.

Parameters

- **fitting_type** (`str`)
  - **map** for the maximum a posteriori fitting;
  - **mle** for the maximum likelihood fitting.
- **possible_states** (`List[str]`) – the possible states of the process.
- **sequence** (`Union[List[int], List[str]]`) – the observed sequence of states.
- **fitting_param** (`Any`) –

  - In the maximum a posteriori fitting, the matrix for the a priori distribution (*if omitted, a default value of 1 is assigned to each matrix element*).
  - In the maximum likelihood fitting, a boolean indicating whether to apply a Laplace smoothing to compensate for the unseen transition combinations (*if omitted, the value is set to True*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

`from_dictionary`(*d*)

The method generates a Markov chain from the given dictionary, whose keys represent state pairs and whose values represent transition probabilities.

Parameters
d (`Dict`[`Tuple`[`str`, `str`], `Union`[`float`, `int`]]) – the dictionary to transform into the transition matrix.

Raises

- *ValidationError* – if any input argument is not compliant.
- `ValueError` – if the transition matrix defined by the dictionary is not valid.

Return Type
MarkovChain

`from_file`(*file_path*)

The method reads a Markov chain from the given file.

Only **csv**, **json**, **txt** and **xml** files are supported; data format is inferred from the file extension.

In **csv** files, data must be structured as follows:

- *Delimiter:* **comma**
- *Quoting:* **minimal**
- *Quote Character:* **double quote**
- *Header Row:* state names
- *Data Rows:* probabilities

In **json** files, data must be structured as an array of objects with the following properties:

- **state_from** *(string)*
- **state_to** *(string)*
- **probability** *(float or int)*

In **txt** files, every line of the file must have the following format:

- **<state_from> <state_to> <probability>**

In **xml** files, the structure must be defined as follows:

- *Root Element:* **MarkovChain**

- *Child Elements:* **Item**, *with attributes:*
  - **state_from** *(string)*
  - **state_to** *(string)*
  - **probability** *(float or int)*

Parameters
file_path (Union[str, Path]) – the location of the file that defines the Markov chain.

Raises
- FileNotFoundError – if the file does not exist.
- OSError – if the file cannot be read or is empty.
- *ValidationError* – if any input argument is not compliant.
- ValueError – if the file contains invalid data.

Return Type
MarkovChain

from_graph(*graph*)

The method generates a Markov chain from the given directed graph, whose transition matrix is obtained through the normalization of edge weights.

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

from_matrix(*m*, *states=None*)

The method generates a Markov chain whose transition matrix is obtained through the normalization of the given matrix.

Parameters
- **m** (Union[ndarray, spmatrix]) – the matrix to transform into the transition matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

gamblers_ruin(*size*, *w*, *states=None*)

The method generates a gambler's ruin Markov chain of given size and win probability.

Parameters
- **size** (int) – the size of the Markov chain.
- **w** (float) – the win probability.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

identity(*size*, *states=None*)

The method generates a Markov chain of given size based on an identity transition matrix.

Parameters

- **size** (int) – the size of the Markov chain.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

population_genetics_model(*model*, *n*, *s=0.0*, *u=1e-09*, *v=1e-09*, *states=None*)

The method generates a Markov chain based on the specified population genetics model.

Parameters

- **model** (str)
  - **moran** for the Moran model;
  - **wright-fisher** for the Wright-Fisher model.
- **n** (int) – the number of individuals.
- **s** (float) – the selection intensity.
- **u** (float) – the backward mutation rate.
- **v** (float) – the forward mutation rate.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

random(*size*, *states=None*, *zeros=0*, *mask=None*, *seed=None*)

The method generates a Markov chain of given size with random transition probabilities.

Notes:

- In the mask parameter, undefined transition probabilities are represented by *NaN* values.

Parameters

- **size** (int) – the size of the Markov chain.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).
- **zeros** (int) – the number of null transition probabilities.

- **mask** (Optional[Union [ndarray, spmatrix]]) – a matrix representing locations and values of fixed transition probabilities.
- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

random_distribution(*size*, *f*, *states=None*, *seed=None*, ***kwargs*)

The method generates a Markov chain of given size using draws from a Numpy random distribution function.

Notes:

- *NaN* values are replaced with zeros
- Infinite values are replaced with finite numbers.
- Negative values are clipped to zero.
- In transition matrix rows with no positive values the states are assumed to be uniformly distributed.
- In light of the above points, random distribution functions must be carefully parametrized.

Parameters

- **size** (int) – the size of the Markov chain.
- **f** (Union[Callable, str]) – the Numpy random distribution function or the name of the Numpy random distribution function.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).
- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.
- ****kwargs** – additional arguments passed to the random distribution function.

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

urn_model(*model*, *n*, *states=None*)

The method generates a Markov chain of size **2N + 1** based on the specified urn model.

Parameters

- **model** (str)
  - **bernoulli-laplace** for the Bernoulli-Laplace urn model;
  - **ehrenfest** for the Ehrenfest urn model.
- **n** (int) – the number of elements in each urn.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
MarkovChain

class pydtmc.**MarkovChain**(*p*, *states=None*)

Defines a Markov chain with the given transition matrix.

Parameters

- **p** (Union[ndarray, spmatrix]) – the transition matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1*).

Raises
ValidationError – if any input argument is not compliant.

CHAPTER

TWO

HIDDEN MARKOV MODEL

2.1 Properties

class pydtmc.**HiddenMarkovModel**(*p*, *e*, *states=None*, *symbols=None*)

Defines a hidden Markov model with the given transition and emission matrices.

Parameters

- **p** (Union[ndarray, spmatrix]) – the transition matrix.
- **e** (Union[ndarray, spmatrix]) – the emission matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1 with prefix P*).
- **symbols** (Optional[List[str]]) – the name of each symbol (*if omitted, an increasing sequence of integers starting at 1 with prefix E*).

Raises

ValidationError – if any input argument is not compliant.

e: ndarray

A property representing the emission matrix of the hidden Markov model.

is_ergodic: bool

A property indicating whether the hidden Markov model is ergodic.

is_regular: bool

A property indicating whether the hidden Markov model is regular.

k: int

A property representing the size of the hidden Markov model symbol space.

n: int

A property representing the size of the hidden Markov model state space.

p: ndarray

A property representing the transition matrix of the hidden Markov model.

size: Tuple[int, int]

A property representing the size of the hidden Markov model.
The first value represents the number of states, the second value represents the number of symbols.

states: List[str]

A property representing the states of the hidden Markov model.

symbols: List[str]

A property representing the symbols of the hidden Markov model.

2.2 Instance Methods

class pydtmc.**HiddenMarkovModel**(*p*, *e*, *states=None*, *symbols=None*)

Defines a hidden Markov model with the given transition and emission matrices.

Parameters

- **p** (Union[ndarray, spmatrix]) – the transition matrix.
- **e** (Union[ndarray, spmatrix]) – the emission matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1 with prefix P*).
- **symbols** (Optional[List[str]]) – the name of each symbol (*if omitted, an increasing sequence of integers starting at 1 with prefix E*).

Raises

ValidationError – if any input argument is not compliant.

decode(*symbols*, *initial_status=None*, *use_scaling=True*)

The method calculates the log probability, the posterior probabilities, the backward probabilities and the forward probabilities of an observed sequence of symbols.

Notes:

- If the observed sequence of symbols cannot be decoded, then None is returned.

Parameters

- **symbols** (Union[List[int], List[str]]) – the observed sequence of symbols.
- **initial_status** (Optional[Union [int, str, ndarray, spmatrix]]) – the initial state or the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).
- **use_scaling** (bool) – a boolean indicating whether to return scaled backward and forward probabilities together with their scaling factors.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Optional[Tuple[float, ndarray, ndarray, ndarray, Optional[ndarray]]]

emission_probability(*symbol*, *state*)

The method computes the probability of a given symbol, conditioned on the process being at a given state.

Parameters

- **symbol** (Union[int, str]) – the target symbol.

- **state** (Union[int, str]) – the origin state.

Raises
ValidationError – if any input argument is not compliant.

Return Type
float

next(*initial_state*, *target='both'*, *output_index=False*, *seed=None*)

The method simulates a single step in a random walk.

Parameters

- **initial_state** (Union[int, str]) – the initial state.
- **target** (str)
  - **state** for a random state;
  - **symbol** for a random symbol;
  - **both** for a random state and a random symbol.
- **output_index** (bool) – a boolean indicating whether to output the state index.
- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Union[int, str, Tuple[int, int], Tuple[str, str]]

predict(*prediction_type*, *symbols*, *initial_status=None*, *output_indices=False*)

The method calculates the log probability and the most probable states path of an observed sequence of symbols.

Notes:

- If the maximum a posteriori prediction is used and the observed sequence of symbols cannot be decoded, then None is returned.
- If the maximum likelihood prediction is used and the observed sequence of symbols produces null transition probabilities, then None is returned.

Parameters

- **prediction_type** (str)
  - **map** for the maximum a posteriori prediction;
  - **mle** or **viterbi** for the maximum likelihood prediction.
- **symbols** (Union[List[int], List[str]]) – the observed sequence of symbols.
- **initial_status** (Optional[Union [int, str, ndarray, spmatrix]]) – the initial state or the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).
- **output_indices** (bool) – a boolean indicating whether to output the state indices.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Tuple[float, Union[List[int], List[str]]]

restrict(*states=None*, *symbols=None*)

The method returns a submodel restricted to the given states and symbols.

Notes:

- Submodel transition and emission matrices are normalized so that their rows sum to 1.0.
- Submodel transition and emission matrices whose rows sum to 0.0 are replaced by uniformly distributed probabilities.

Parameters

- **states** (Optional[Union [int, str, List[int], List[str]]]) – the states to include in the submodel.
- **symbols** (Optional[Union [int, str, List[int], List[str]]]) – the symbols to include in the submodel.

Raises
ValidationError – if any input argument is not compliant.

Return Type
HiddenMarkovModel

simulate(*steps*, *initial_state=None*, *final_state=None*, *final_symbol=None*, *output_indices=False*, *seed=None*)

The method simulates a random sequence of states and symbols of the given number of steps.

Parameters

- **steps** (int) – the number of steps.
- **initial_state** (Optional[Union [int, str]]) – the initial state (*if omitted, it is chosen uniformly at random*).
- **final_state** (Optional[Union [int, str]]) – the final state of the simulation (*if specified, the simulation stops as soon as it is reached even if not all the steps have been performed*).
- **final_symbol** (Optional[Union [int, str]]) – the final state of the simulation (*if specified, the simulation stops as soon as it is reached even if not all the steps have been performed*).
- **output_indices** (bool) – a boolean indicating whether to output the state indices.
- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Union[Tuple[List[int], List[int]], Tuple[List[str], List[str]]]

to_dictionary()

The method returns a dictionary representing the hidden Markov model.

Return Type
Dict[Tuple[str, str, str], float]

`to_file`(*file_path*)

The method writes a hidden Markov model to the given file.

Only **csv**, **json**, **txt** and **xml** files are supported; data format is inferred from the file extension.

Parameters

file_path (Union[str, Path]) – the location of the file in which the hidden Markov model must be written.

Raises

- OSError – if the file cannot be written.
- *ValidationError* – if any input argument is not compliant.

`to_graph`()

The method returns a directed graph representing the hidden Markov model.

Return Type

DiGraph

`to_matrices`()

The method returns a tuple of two items representing the underlying matrices of the hidden Markov model.

The first item is the transition matrix and the second item is the emission matrix.

Return Type

Tuple[ndarray, ndarray]

`transition_probability`(*state_target*, *state_origin*)

The method computes the probability of a given state, conditioned on the process being at a given state.

Parameters

- **state_target** (Union[int, str]) – the target state.
- **state_origin** (Union[int, str]) – the origin state.

Raises

ValidationError – if any input argument is not compliant.

Return Type

float

2.3 Static Methods

`class` pydtmc.`HiddenMarkovModel`(*p*, *e*, *states=None*, *symbols=None*)

Defines a hidden Markov model with the given transition and emission matrices.

Parameters

- **p** (Union[ndarray, spmatrix]) – the transition matrix.
- **e** (Union[ndarray, spmatrix]) – the emission matrix.

- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1 with prefix P*).
- **symbols** (Optional[List[str]]) – the name of each symbol (*if omitted, an increasing sequence of integers starting at 1 with prefix E*).

Raises
ValidationError – if any input argument is not compliant.

estimate(*possible_states*, *possible_symbols*, *sequence_states*, *sequence_symbols*)

The method performs the maximum likelihood estimation of transition and emission probabilities from an observed sequence of states and symbols.

Parameters

- **possible_states** (List[str]) – the possible states of the model.
- **possible_symbols** (List[str]) – the possible symbols of the model.
- **sequence_states** (Union[List[int], List[str]]) – the observed sequence of states.
- **sequence_symbols** (Union[List[int], List[str]]) – the observed sequence of symbols.

Raises
ValidationError – if any input argument is not compliant.

Return Type
HiddenMarkovModel

fit(*fitting_type*, *possible_states*, *possible_symbols*, *p_guess*, *e_guess*, *symbols*, *initial_status=None*)

The method fits a hidden Markov model from an initial guess and one or more observed sequences of symbols.

Parameters

- **fitting_type** (str)
  - **baum-welch** for the Baum-Welch fitting;
  - **map** for the maximum a posteriori fitting;
  - **mle** or **viterbi** for the maximum likelihood fitting.
- **possible_states** (List[str]) – the possible states of the model.
- **possible_symbols** (List[str]) – the possible symbols of the model.
- **p_guess** (ndarray) – the initial transition matrix guess.
- **e_guess** (ndarray) – the initial emission matrix guess.
- **symbols** (Union[List[int], List[str], List[List[int]], List[List[str]]]) – the observed sequence(s) of symbols.
- **initial_status** (Optional[Union [int, str, ndarray, spmatrix]]) – the initial state or the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).

Raises

- *ValidationError* – if any input argument is not compliant.
- ValueError – if the fitting algorithm fails to converge.

Return Type
HiddenMarkovModel

from_dictionary(*d*)

The method generates a hidden Markov model from the given dictionary, whose keys represent state pairs and whose values represent transition probabilities.

Parameters
d (Dict[Tuple[str, str, str], Union[float, int]]) – the dictionary to transform into the transition matrix.

Raises

- *ValidationError* – if any input argument is not compliant.
- ValueError – if the transition matrix defined by the dictionary is not valid.

Return Type
HiddenMarkovModel

from_file(*file_path*)

The method reads a hidden Markov model from the given file.

Only **csv**, **json**, **txt** and **xml** files are supported; data format is inferred from the file extension.
Transition probabilities are associated to reference attribute "P", emission probabilities are associated to reference attribute "E".

In **csv** files, data must be structured as follows:

- *Delimiter:* **comma**
- *Quoting:* **minimal**
- *Quote Character:* **double quote**
- *Header Row:* state names (prefixed with "P_") and symbol names (prefixed with "E_")
- *Data Rows:* **probabilities**

In **json** files, data must be structured as an array of objects with the following properties:

- **reference** *(string)*
- **element_from** *(string)*
- **element_to** *(string)*
- **probability** *(float or int)*

In **txt** files, every line of the file must have the following format:

- **<reference> <element_from> <element_to> <probability>**

In **xml** files, the structure must be defined as follows:

- *Root Element:* **HiddenMarkovModel**
- *Child Elements:* **Item**, *with attributes:*
  - **reference** *(string)*
  - **element_from** *(string)*
  - **element_to** *(string)*
  - **probability** *(float or int)*

Parameters
file_path (Union[str, Path]) – the location of the file that defines the hidden Markov model.

Raises
- FileNotFoundError – if the file does not exist.
- OSError – if the file cannot be read or is empty.
- *ValidationError* – if any input argument is not compliant.
- ValueError – if the file contains invalid data.

Return Type
HiddenMarkovModel

from_graph(*graph*)

The method generates a hidden Markov model from the given directed graph, whose transition and emission matrices are obtained through the normalization of edge weights.

Raises
ValidationError – if any input argument is not compliant.

Return Type
HiddenMarkovModel

from_matrices(*mp*, *me*, *states=None*, *symbols=None*)

The method generates a hidden Markov model whose transition and emission matrices are obtained through the normalization of the given matrices.

Parameters
- **mp** (Union[ndarray, spmatrix]) – the matrix to transform into the transition matrix.
- **me** (Union[ndarray, spmatrix]) – the matrix to transform into the emission matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1 with prefix P*).
- **symbols** (Optional[List[str]]) – the name of each symbol (*if omitted, an increasing sequence of integers starting at 1 with prefix E*).

Raises
ValidationError – if any input argument is not compliant.

Return Type
HiddenMarkovModel

random(*n*, *k*, *states=None*, *p_zeros=0*, *p_mask=None*, *symbols=None*, *e_zeros=0*, *e_mask=None*, *seed=None*)

The method generates a Markov chain of given size with random transition probabilities.

Notes:

- In the mask parameter, undefined transition probabilities are represented by *NaN* values.

Parameters

- **n** (int) – the number of states.
- **k** (int) – the number of symbols.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1 with prefix P*).
- **p_zeros** (int) – the number of null transition probabilities.
- **p_mask** (Optional[Union [ndarray, spmatrix]]) – a matrix representing locations and values of fixed transition probabilities.
- **symbols** (Optional[List[str]]) – the name of each symbol (*if omitted, an increasing sequence of integers starting at 1 with prefix E*).
- **e_zeros** (int) – the number of null emission probabilities.
- **e_mask** (Optional[Union [ndarray, spmatrix]]) – a matrix representing locations and values of fixed emission probabilities.
- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.

Raises

ValidationError – if any input argument is not compliant.

Return Type

HiddenMarkovModel

class pydtmc.**HiddenMarkovModel**(*p*, *e*, *states=None*, *symbols=None*)

Defines a hidden Markov model with the given transition and emission matrices.

Parameters

- **p** (Union[ndarray, spmatrix]) – the transition matrix.
- **e** (Union[ndarray, spmatrix]) – the emission matrix.
- **states** (Optional[List[str]]) – the name of each state (*if omitted, an increasing sequence of integers starting at 1 with prefix P*).
- **symbols** (Optional[List[str]]) – the name of each symbol (*if omitted, an increasing sequence of integers starting at 1 with prefix E*).

Raises

ValidationError – if any input argument is not compliant.

CHAPTER

THREE

ASSESSMENT FUNCTIONS

pydtmc.**assess_first_order**(*possible_states*, *sequence*, *significance=0.05*)

The function verifies whether the given sequence can be associated to a first-order Markov process.

Parameters

- **possible_states** (List[str]) – the possible states of the process.
- **sequence** (Union[List[int], List[str]]) – the observed sequence of states.
- **significance** (float) – the p-value significance threshold below which to accept the alternative hypothesis.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Tuple[Optional[bool], float, Dict[str, Any]]

pydtmc.**assess_homogeneity**(*possible_states*, *sequences*, *significance=0.05*)

The function verifies whether the given sequences belong to the same Markov process.

Parameters

- **possible_states** (List[str]) – the possible states of the process.
- **sequences** (List[Union[List[int], List[str]]]) – the observed sequences of states.
- **significance** (float) – the p-value significance threshold below which to accept the alternative hypothesis.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Tuple[Optional[bool], float, Dict[str, Any]]

pydtmc.**assess_markov_property**(*possible_states*, *sequence*, *significance=0.05*)

The function verifies whether the given sequence holds the Markov property.

Parameters

- **possible_states** (List[str]) – the possible states of the process.
- **sequence** (Union[List[int], List[str]]) – the observed sequence of states.
- **significance** (float) – the p-value significance threshold below which to accept the alternative hypothesis.

Raises

ValidationError – if any input argument is not compliant.

Return Type
Tuple[Optional[bool], float, Dict[str, Any]]

pydtmc.**assess_stationarity**(*possible_states*, *sequence*, *blocks=1*, *significance=0.05*)

The function verifies whether the given sequence is stationary.

Parameters

- **possible_states** (List[str]) – the possible states of the process.
- **sequence** (Union[List[int], List[str]]) – the observed sequence of states.
- **blocks** (int) – the number of blocks in which the sequence is divided.
- **significance** (float) – the p-value significance threshold below which to accept the alternative hypothesis.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Tuple[Optional[bool], float, Dict[str, Any]]

pydtmc.**assess_theoretical_compatibility**(*mc*, *sequence*, *significance=0.05*)

The function verifies whether the given empirical sequence is statistically compatible with the given theoretical Markov process.

Parameters

- **mc** (*MarkovChain*) – a Markov chain representing the theoretical process.
- **sequence** (Union[List[int], List[str]]) – the observed sequence of states.
- **significance** (float) – the p-value significance threshold below which to accept the alternative hypothesis.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Tuple[Optional[bool], float, Dict[str, Any]]

CHAPTER

FOUR

PLOTTING FUNCTIONS

pydtmc.**plot_comparison**(*models*, *underlying_matrices='transition'*, *names=None*, *dpi=100*)

The function plots the underlying matrices of the given models in the form of a heatmap.

Notes:

- If Matplotlib is in interactive mode, the plot is immediately displayed and the function does not return the plot handles.

Parameters

- **models** (List[Union[*HiddenMarkovModel*, *MarkovChain*]]) – the models.
- **underlying_matrices** (str)
  - **emission** for comparing the emission matrices;
  - **transition** for comparing the transition matrices.
- **names** (Optional[List[str]]) – the name of each model subplot (*if omitted, a standard name is given to each subplot*).
- **dpi** (int) – the resolution of the plot expressed in dots per inch.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Optional[Tuple[Figure, Union[Axes, List[Axes]]]]

pydtmc.**plot_eigenvalues**(*model*, *dpi=100*)

The function plots the eigenvalues of the transition matrix of the given model on the complex plane.

Notes:

- If Matplotlib is in interactive mode, the plot is immediately displayed and the function does not return the plot handles.

Parameters

- **model** (Union[*HiddenMarkovModel*, *MarkovChain*]) – the model.
- **dpi** (int) – the resolution of the plot expressed in dots per inch.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Optional[Tuple[Figure, Union[Axes, List[Axes]]]]

pydtmc.**plot_flow**(*model*, *steps*, *interval*, *initial_status=None*, *palette='viridis'*, *dpi=100*)

The function produces an alluvial diagram of the given model.

Notes:

- If Matplotlib is in interactive mode, the plot is immediately displayed and the function does not return the plot handles.

Parameters

- **model** (Union[*HiddenMarkovModel*, *MarkovChain*]) – the model.
- **steps** (int) – the number of steps.
- **interval** (int) – the interval between each step.
- **initial_status** (Optional[Union [int, str, ndarray, spmatrix]]) – the initial state or the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).
- **palette** (str) – the palette of the plot.
- **dpi** (int) – the resolution of the plot expressed in dots per inch.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Optional[Tuple[Figure, Union[Axes, List[Axes]]]]

pydtmc.**plot_graph**(*model*, *nodes_color=True*, *nodes_shape=True*, *edges_label=True*, *force_standard=False*, *dpi=100*)

The function plots the directed graph of the given model.

Notes:

- If Matplotlib is in interactive mode, the plot is immediately displayed and the function does not return the plot handles.
- Graphviz and pydot are not required, but they provide access to extended mode with improved rendering and additional features.
- The rendering, especially in standard mode or for big graphs, is not granted to be high-quality.
- For Markov chains, the color of nodes is based on communicating classes; for hidden Markov models, every state node has a different color and symbol nodes are gray.
- For Markov chains, recurrent nodes have an elliptical shape and transient nodes have a rectangular shape; for hidden Markov models, state nodes have an elliptical shape and symbol nodes have a hexagonal shape.

Parameters

- **model** (Union[*HiddenMarkovModel*, *MarkovChain*]) – the model.
- **nodes_color** (bool) – a boolean indicating whether to use a different color for every type of node.
- **nodes_shape** (bool) – a boolean indicating whether to use a different shape for every type of node.
- **edges_label** (bool) – a boolean indicating whether to display the probability of every edge as text.
- **force_standard** (bool) – a boolean indicating whether to use standard mode even if extended mode is available.
- **dpi** (int) – the resolution of the plot expressed in dots per inch.

Raises
ValidationError – if any input argument is not compliant.

Return Type
Optional[Tuple[Figure, Union[Axes, List[Axes]]]]

pydtmc.**plot_redistributions**(*model*, *redistributions*, *initial_status=None*, *plot_type='projection'*, *dpi=100*)

The function plots a redistribution of states on the given model.

Notes:

- If Matplotlib is in interactive mode, the plot is immediately displayed and the function does not return the plot handles.

Parameters

- **model** (Union[*HiddenMarkovModel*, *MarkovChain*]) – the model to be converted into a graph.
- **redistributions** (int) – the number of redistributions to perform.
- **initial_status** (Optional[Union [int, str, ndarray, spmatrix]]) – the initial state or the initial distribution of the states (*if omitted, the states are assumed to be uniformly distributed*).
- **plot_type** (str)
  - **heatmap** for displaying a heatmap plot;
  - **projection** for displaying a projection plot.
- **dpi** (int) – the resolution of the plot expressed in dots per inch.

Raises

- *ValidationError* – if any input argument is not compliant.
- ValueError – if the "distributions" parameter represents a sequence of redistributions and the "initial_status" parameter does not match its first element.

Return Type
Optional[Tuple[Figure, Union[Axes, List[Axes]]]]

pydtmc.**plot_sequence**(*model*, *steps*, *initial_state=None*, *plot_type='histogram'*, *seed=None*, *dpi=100*)

The function plots a random walk on the given model.

Notes:

- If Matplotlib is in interactive mode, the plot is immediately displayed and the function does not return the plot handles.

Parameters

- **model** (Union[*HiddenMarkovModel*, *MarkovChain*]) – the model.
- **steps** (int) – the number of steps.
- **initial_state** (Optional[Union [int, str]]) – the initial state of the random walk (*if omitted, it is chosen uniformly at random*).
- **plot_type** (str)
  - **heatmap** for displaying heatmap-like plots;
  - **histogram** for displaying a histogram plots;
  - **matrix** for displaying matrix plots.
- **seed** (Optional[int]) – a seed to be used as RNG initializer for reproducibility purposes.
- **dpi** (int) – the resolution of the plot expressed in dots per inch.

Raises

ValidationError – if any input argument is not compliant.

Return Type

Optional[Tuple[Figure, Union[Axes, List[Axes]]]]

CHAPTER

FIVE

CUSTOM EXCEPTIONS

exception pydtmc.**ValidationError**

Defines an exception thrown when inappropriate argument values are provided.

CHAPTER

SIX

INDEX

PyDTMC is a full-featured and lightweight library for discrete-time Markov chains analysis.
It provides classes and functions for creating, manipulating, simulating and visualizing Markov processes.

Current Version: 8.7.0

INDEX