STP

Release 2019

Jonathan Delgado

Mar 24, 2023

CONTENTS:

CHAPTER

ONE

STP

Handles generating random objects to use in various calculations.

Author: Jonathan Delgado

stp.stochastic.**KMC**(*W*, *p*, *num_paths*, *path_length*, *time_step=1*, *seed=None*, *_degenerate_threshold=0.985*)

A Rejection-free Kinetic Monte Carlo (KMC) algorithm for simulating the discrete time evolution of a system, where some processes can occur with known (continuous time) rates W = W(t). The discrete time dynamics will be computed using the continuous time rate matrix, W, by freezing the control parameter for moments of time, time_step. From: https://en.wikipedia.org/wiki/Kinetic_Monte_Carlo.

Args:

W (np.ndarray/function): rate matrix for np.ndarray. If function is provided then W is the function of time that provides a rate matrix (W = W(t)) for each moment in time. Will be used to implement driven systems. If W is just a rate matrix then W(t) is just the time-homogeneous rate matrix.

p (np.ndarray): the initial marginal distribution.

num_paths (int): the number of paths to sample.

path_length (int): the length of each path.

Kwargs:

time_step (float): the interval between changing of rate matrix (changing of control parameter), and the delay between observations.

seed (None/int): the seed for sampling. Unseeded (uses module-level rng) if None.

_degenerate_threshold (float): the fraction of paths that we require to be unique. If not satisfied KMC will be run again with an increased threshold.

Returns:

(np.ndarray): matrix of paths sampled via KMC.

stp.stochastic.**complete_path_space**(*n*, *path_length*)

Generates the entire path space as a matrix with each row corresponding to a path and each column corresponding to an observation step.

Args:

n (int): the number of states in the state space

path_length (int): the length of each path

Returns:

(np.ndarray): the entire path space. Will be a matrix of size: n^path_length x path_length.

stp.stochastic.**direct_sampling**(*R*, *p*)

Not yet been implemented: Samples the path space directly to reflect the dynamics given by the initial marginal and the transition matrix.

Args:
R (function/np.ndarray): the (possibly time-dependent) transition matrix.

p (np.ndarray): the marginal distribution.

Returns:
(np.ndarray): the sampled portion of the path space.

stp.stochastic.**get_path_probability**(*R*, *p*, *path*)

Calculates the probability of observing a provided path using the a (potentially time-inhomogeneous) transition matrix, the initial marginal distribution, and the path itself.

Args:
R (function/np.ndarray): the transition matrix. A function of the observation step if the matrix is time-inhomogeneous. That is, R = R(n). In which case Pr(x <- y) = R(1)[x,y] * p[y]. Provide a numpy matrix in the case of a time-homogeneous transition matrix.

p (np.ndarray): the marginal distribution.

path (list/np.ndarray): the path.

Returns:
(float): the probability of observing this path.

stp.stochastic.**get_stationary_distribution**(*matrix*, *discrete=True*)

Calculates the stationary distribution of a transition or rate matrix. Credit: https://stackoverflow.com/questions/31791728/python-code-explanation-for-stationary-distribution-of-a-markov-chain.

Args:
matrix (np.ndarray/function): the transition or rate matrix

Kwargs:
discrete (bool): True if the provided matrix is a discrete time transition matrix, False if the provided matrix is a continuous time rate matrix

Returns:
(np.ndarray/function): the limiting distribution (as a function of time in the case where the matrix is too)

stp.stochastic.**rand_p**(*n=2*, *zeros=0*)

Creates a random probability distribution, currently implemented only with uniform sampling.

Kwargs:
n (int): the dimension of the desired distribution

zeros (int): the number of desired zeros to be injected into the distribution. Allows for ease of testing edge cases.

Returns:
(np.ndarray): the random distribution

stp.stochastic.**rand_rate_matrix**(*n=2*, *seed=None*)

Generates a random, time-independent, n x n rate matrix consisting of probabilities per unit time. Column normalized.

Kwargs:
n (int): the number of states the matrix will correspond to. Relates to the dimensions of the matrix.

seed (None/int): the seed for sampling. Unseeded (uses module-level rng) if None.

Returns:
(np.ndarray): the rate matrix.

stp.stochastic.**rand_transition_matrix**(*n=2*, *time_step=1.0*)

Generates a random, time-independent, discrete time, transition matrix by first generating a random rate matrix and then matrix exponentiating it to incorporate the time step as an additional parameter.

Kwargs:

n (int): the number of states the matrix will correspond to. Relates to the dimensions of the matrix.

time_step (float): the time step: the interval of time between observations.

Returns:

(np.ndarray): the n x n transition matrix

stp.stochastic.**rate_to_transition_matrix**(*W*, *time_step*)

Converts a rate matrix to a transition matrix assuming a constant control parameter during the duration of time_step.

Args:

W (np.ndarray): the rate matrix

time_step (float): the time step

Returns:

(np.ndarray): the transition matrix

stp.stochastic.**self_assembly_rate_matrix**(*alpha=1*, *c=1*, *M=1*)

Generates a self-assembly rate matrix for a 3-state system following: https://aip.scitation.org/doi/10.1063/1.3662140.

Kwargs:

alpha (float/function): the energy/temperature coupled parameter as a float or a function which returns alpha as a function of time. In the latter case the returned rate matrix will be a function which provides a np.ndarray as a function of time. The energy is the negative of the "optimally bound level of energy".

c (float): "concentration-like variable".

M (int): the degeneracy of the misbound level.

Returns:

(np.ndarray/function): the time-independent rate matrix as a numpy array in the case where the temperature is constant. Otherwise returns a function corresponding to the time-dependent rate matrix.

stp.stochastic.**self_assembly_transition_matrix**(*alpha=1*, *c=1*, *M=1*, *time_step=1*)

Generates a self-assembly, discrete time, transition matrix for a 3-state system following: https://aip.scitation.org/doi/10.1063/1.3662140. Done assuming any external control parameter is fixed for the duration of the time_step. This matrix is step dependent, so conversions will be done to time to calculate the current temperature.

Kwargs:

alpha (float/function): the energy/temperature coupled parameter as a float or a function which returns alpha as a function of time. In the latter case the returned transition matrix will be a function which provides a np.ndarray as a function of time. The energy is the negative of the "optimally bound level of energy". This external control parameter will be fixed for the duration of the time_step.

c (float): "concentration-like variable"

M (int): the degeneracy of the misbound level

time_step (float): the time step

Returns:

(np.ndarray/function): the transition matrix

stp.stochastic.**step**(*matrix*, *p*)

Evolves a probability distribution one step forward by computing the matrix multiplication between matrix and p. In the case of the matrix being a rate matrix the output is the time-derivative of p.

Args:

matrix (np.ndarray): the transition or rate matrix

p (np.ndarray): the marginal distribution

Returns:

(np.ndarray): the evolved marginal

CHAPTER

TWO

STP INFORMATION

Entropy and information theory related calculations.

Author: Jonathan Delgado

`class stp.info.InfoSpace(`*paths*, *p_matrix*`)`

Information space. Holds collections of paths that traverse states in a state space as a matrix, and the probability of each of those paths.

Provides functionality on this path space such as providing path entropies.

Attributes:

paths: the matrix of paths.

probabilities: a list of probabilities each path.

num_paths: the number of paths considered.

path_length: the length of the paths considered.

probabilities: a matrix where the (i,j)th element is the probability of observing the first j states of the ith path.

entropies: a list of path entropies for each path

total_probability: the sum of the probabilities of each path.

`property entropies`

Returns a list of path entropies for each corresponding path probability.

`static shorten(`*infospace*, *path_length*, *return_index=False*`)`

Takes an Information Space and shortens it. Since unique paths of length n, may be degenerate when truncated to paths of length m < n, we need to check for degeneracies and filter them out in both paths and probabilities.

Args:

infospace (InfoSpace): the information space to shorten.

path_length (int): the path length the information space should be shortened to.

Kwargs:

return_index (bool): returns the indices of the non-degenerate paths for the given path length using the original matrix. Useful for filtering other quantities of interest that may not be attached to this object.

Returns:

(InfoSpace): the shortened InfoSpace.

class stp.info.**PartitionedInfoSpace**(*entropy_rates*, *epsilon*, *paths=None*, *p_matrix=None*, *typical_space=None*, *atypical_space=None*)

Partitioned Information Space. Constructs a typical set on an information space to partition it into a typical information space and an atypical one.

Holds path probabilities, typical paths, atypical paths, atypical path probabilities and more. This object will use a provided (often sampled) path space to partition the space into a collection of typical and atypical paths depending on the dynamics provided. Will also track other quantities of interest such as the upper and lower bounds on the path probabilities required for the paths to be considered typical.

Attributes:

paths: the matrix of paths.

probabilities: a list of probabilities each path.

num_paths: the number of paths considered.

path_length: the length of the paths considered.

probabilities: a matrix where the (i,j)th element is the probability of observing the first j states of the ith path.

entropies: a list of path entropies for each path.

entropy_rates: a list of the entropy rates for each various path length. This will be the center of the epsilon-neighborhood for path entropies to qualify paths as typical for.

epsilon: the widths of the neighborhood used for paths to be considered typical for each path length.

upper/lower: the upper/lower bounds as measured in nats. This means that a path is typical if and only if its path entropy rate is within these bounds.

typicalities: a matrix where the (i,j)th element is a boolean determining whether the ith path is typical after j+1 steps.

ts: the typical set.

ats: the atypical set.

static partition_space(*R*, *p*, *paths*, *epsilon=0.5*, *return_p=False*)

Partitions a path space using the dynamics provided.

Args:

R (np.ndarray/function): the transition matrix, time-dependent if provided as a function.

p (np.ndarray): the initial marginal distribution.

paths (np.ndarray): the portion of the path space to use.

Kwargs:

epsilon (float/np.ndarray): the radius/radii of the epsilon neighborhood to consider paths to be typical within.

return_p (bool): False, return only the PartitionedInfoSpace, True returns both the PartitionedInfoSpace and a list of the marginal vs time.

Returns:

(ParitionedInfoSpace/2-tuple): the PartitionedInfoSpace (PIS) or the PIS and a list of the marginal versus observation step if return_p is True.

static shorten(*pinfospace*, *path_length*, *return_index=False*)

Takes a PartitionedInformationSpace and shortens it. Since unique paths of length n, may be degenerate when truncated to paths of length m < n, we need to check for degeneracies and filter them out in both paths and probabilities.

Args:
pinfospace (PartitionedInfoSpace): the partitioned information space to shorten.

path_length (int): the path length the information space should be shortened to.

Kwargs:
return_index (bool): returns the indices of the non-degenerate paths for the given path length using the original matrix. Useful for filtering other quantities of interest that may not be attached to this object.

Returns:
(PartitionedInfoSpace): the shortened PartitionedInfoSpace.

property typicalities

Returns the matrix of typicalities.

stp.info.**delta_entropy**(*R*, *p*)

Calculates the discrete time change in entropy using the entropy of p evolved with R, minus the entropy of p.

Args:
R (np.ndarray): the transition matrix.

p (np.ndarray): the marginal distribution.

Returns:
(float): the change in entropy

stp.info.**entropy**(*p*)

Calculates the Shannon entropy for a marginal distribution.

Args:
p (np.ndarray): the marginal distribution.

Returns:
(float): the entropy of p

stp.info.**entropy_flow**(*R*, *p*)

Calculates the discrete time entropy flow. This has not been generalized to handle the continuous time entropy flow yet.

Args:
R (np.ndarray): the discrete time transition matrix

p (np.ndarray): the marginal distribution

Returns:
(float): the entropy flow

stp.info.**entropy_production**(*matrix*, *p*, *discrete=True*)

Calculates the entropy production for either discrete or continuous time.

Args:
matrix (np.ndarray): the stochastic matrix, either a discrete time transition matrix or a continuous time rate matrix.

p (np.ndarray): the marginal distribution

Kwargs:
discrete (bool): True if we are calculating the discrete time entropy production (nats), False if we are calculating it in continuous time (nats/time).

Returns:
(float/np.inf): the entropy production

stp.info.**entropy_rate**(*R*)

Calculates the asymptotic entropy rate for the provided transition matrix. If the matrix is time-inhomogeneous then we return a function that generates the entropy_rate as a function of n by calculating the systems limiting distribution for each n.

Args:
R (np.ndarray/function): the transition matrix.

Returns:
(float/function): the entropy velocity.

stp.info.**relative_entropy**(*p*, *q*)

Calculates the Kullback-Leibler divergence, which is nonnegative and vanishes if and only if the distributions coincide.

Args:
p, q (np.ndarray): the probability distributions.

Returns:
(float): the relative entropy.

CHAPTER

THREE

STP GUI

GUI element handler.

Author: Jonathan Delgado

Handles creating GUI elements such a graphical loading bars for long processes.

class stp.tools.gui.**ProgressBar**(*MAX_VALUE*, *width=200*, *height=40*, *title='Loading...'*)

Tkinter progress bar.

Object which will handle creating a Tkinter progress window for the purpose of showing as a graphical loading bar.

Attributes:

MAX_VALUE: the max value of steps before completing the bar.

width: the width of the GUI bar.

height: the height of the GUI bar.

title: the window title.

finish()

Finishes the bar. Briefly shows it has been completed before safely destroying the Tkinter window.

next()

Mask for update to use same syntax in the case of using instead of ShadyBar for example.

set_title(*title*)

Updates the bar's title.

Args:

title (str): the new title.

Returns:

(None): none

update(*amount=1*)

Main function for interacting with the bar. Handles updating progress.

Kwargs:

amount (int): the amount to update the progress by.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX