SMCPy Documentation

Release 1.0

Patrick Leser

Oct 29, 2018

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Uncertainty quantification (UQ) is necessary to provide meaningful and reliable predictions of real-world system performance. One major obstacle for the implementation of statistical methods for UQ is the use of expensive computational models. Classical UQ methods such as Markov chain Monte Carlo (MCMC) generally require thousands to millions of model evaluations, and, when coupled with an expensive model, result in excessive solve times that can render the analysis intractable. These methods are also inherently serial, which prohibits speedup by high performance computing. Recently, Sequential Monte Carlo (SMC) has emerged as a alternative to MCMC. In contrast, this method uses parallel model evaluations to realize significant speedup.

This software is an implementation of SMC that uses the Message Passing Interface (MPI) to provide users general access to parallel UQ methods in Python 2.7. The algorithm used is based on the work by Nguyen et al. ["Efficient Sequential Monte-Carlo Samplers for Bayesian Inference" IEEE Transactions on Signal Processing, Vol. 64, No. 5 (2016)]. To operate the code, the user supplies a computational model built in Python 2.7, defines prior distributions for each of the model parameters to be estimated, and provides data to be used for calibration. SMC sampling can then be conducted with ease through instantiation of the SMC class and a call to the sample() method. The output of this process is an approximation of the parameter posterior probability distribution conditioned on the data provided.

CHAPTER

TWO

SOURCE CODE DOCUMENTATION

Documentation for the primary SMCPy classes.

2.1 SMC Module Documentation

class smcpy.**SMCSampler**(*data*, *model*, *param_priors*)

Class for performing parallel Sequential Monte Carlo sampling

load_particle_chain(*h5_file*)

Loads and returns a particle chain object stored using the HDF5Storage class.

Parameters **hdf5_to_load** (*string*) – file path of a particle chain saved using the ParticleChain.save() or self.save_particle_chain() methods.

sample(*num_particles*, *num_time_steps*, *num_mcmc_steps*, *measurement_std_dev*, *ESS_threshold=None*, *proposal_center=None*, *proposal_scales=None*, *restart_time_step=0*, *hdf5_to_load=None*, *autosave_file=None*)

Parameters

- **num_particles** (*int*) – number of particles to use during sampling
- **num_time_steps** (*int*) – number of time steps in temperature schedule that is used to transition between prior and posterior distributions.
- **num_mcmc_steps** – number of mcmc steps to take during mutation
- **num_mcmc_steps** – int
- **measurement_std_dev** (*float*) – standard deviation of the measurement error
- **ESS_threshold** (*float or int*) – threshold equivalent sample size; triggers resampling when ESS > ESS_threshold
- **proposal_center** (*dict*) – initial parameter dictionary, which is used to define the initial proposal distribution when generating particles; default is None, and initial proposal distribution = prior.
- **proposal_scales** (*dict*) – defines the scale of the initial proposal distribution, which is centered at proposal_center, the initial parameters; i.e. prop ~ MultivarN(q1, (I*proposal_center*scales)^2). Proposal scales should be passed as a dictionary with keys and values corresponding to parameter names and their associated scales, respectively. The default is None, which sets initial proposal distribution = prior.
- **restart_time_step** (*int*) – time step at which to restart sampling; default is zero, meaning the sampling process starts at the prior distribution; note that restart_time_step <

num_time_steps. The step at restart_time is retained, and the sampling begins at the next step (t=restart_time_step+1).

- **hdf5_to_load** (*string*) – file path of a particle chain saved using the ParticleChain.save() method.

Returns A ParticleChain class instance that stores all particles and their past generations at every time step.

save_particle_chain(*h5_file*)
Saves self.particle_chain to an hdf5 file using the HDF5Storage class.

Parameters **hdf5_to_load** (*string*) – file path at which to save particle chain

2.2 Particle Module Documentation

class smcpy.particles.**Particle**(*params*, *weight*, *log_like*)
Class defining data structure of an SMC particle (a member of an SMC particle chain).

Parameters

- **params** (*dictionary*) – parameters associated with particle; keys = parameter name and values = parameter value.
- **weight** (*float*) – the computed weight of the particle
- **log_like** (*float*) – the log likelihood of the particle

copy()
Returns a deep copy of self.

print_particle_info()
Prints particle parameters, weight, and log likelihood to screen.

class smcpy.particles.**ParticleChain**
Class defining data structure of an SMC particle chain. The chain stores all particle instances at all temperature steps (i.e., the entire particle chain is M x T where M is the total number of particles and T is the number of steps in the temperature schedule).

add_particle(*particle*, *step_number*)
Add a single particle to a given step.

add_step(*particle_list*)
Add an entire step to the chain, providing a list of particles.

calculate_step_covariance(*step=-1*)
Estimates the covariance matrix for a given step in the chain.

Parameters **step** (*int*) – step identifier, default is most recent (i.e., step=-1)

compute_ESS(*step=-1*)
Computes the effective sample size (ESS) of a given step in the particle chain.

copy_step(*step=-1*)
Returns a copy of particle chain at step (most recent step by default).

get_likes(*step=-1*)
Returns a list of all particle likelihoods for a given step.

get_log_likes(*step=-1*)
Returns a list of all particle log likelihoods for a given step.

get_mean(*step=-1*)
Computes mean for a given step.

get_num_steps()
Returns number of steps in the particle chain.

get_param_dicts(*step=-1*)
Returns a list of particle parameter dictionaries.

get_params(*key*, *step=-1*)
Returns an array of param <key> with length = number of particles.

get_particles(*step=-1*)
Returns a list of all particles for a given step.

get_weights(*step=-1*)
Returns a list of all weights for a given step.

normalize_step_weights(*step=-1*)
Normalizes weights for all particles in a given step.

Parameters **step** (*int*) – step identifier, default is most recent (i.e., step=-1)

overwrite_step(*step*, *particle_list*)
Overwrite an entire step of the chain with the provided list of particles.

plot_all_marginals(*step=-1*, *save=False*, *show=True*, *prefix='marginal_'*)
Plots marginal approximation for all parameters in the chain.

plot_marginal(*key*, *step=-1*, *save=False*, *show=True*, *prefix='marginal_'*)
Plots a single marginal approximation for param given by <key>.

plot_pairwise_weights(*step=-1*, *param_names=None*, *labels=None*, *save=False*, *show=True*, *param_lims=None*, *label_size=None*, *tick_size=None*, *nbins=None*, *prefix='pairwise'*)
Plots pairwise distributions of all parameter combos. Color codes each by weight.

resample(*step=-1*, *overwrite=True*)
Resamples a given step in the particle chain based on normalized weights. Assigns discrete probabilities to each particle (sum to 1), resample from this discrete distribution using the particle's copy() method.

Parameters **overwrite** (*boolean*) – if True (default), overwrites current step with resampled step, else appends new step

2.3 MCMC Module Documentation

class smcpy.mcmc.**MCMCSampler**(*data*, *model*, *params*, *working_dir='./'*, *storage_backend='pickle'*)
Class for MCMC sampling; based on PyMC. Uses Bayesian inference, MCMC, and a model to estimate parameters with quantified uncertainty based on a set of observations.

Set data and model class member variables, set working directory, and choose storage backend.

Parameters

- **data** (*array_like*) – data to use to inform MCMC parameter estimation; should be same type/shape/size as output of model.
- **model** (*object*) – model for which parameters are being estimated via MCMC; should return output in same type/shape/size as data. A baseclass exists in the Model module that is recommended to define the model object; i.e., model.__bases__ == <class model.Model.Model>,)

- **params** (*dict*) – map where keys are the unknown parameter names (string) and values are lists that define the prior distribution of the parameter [dist name, dist. arg #1, dist. arg #2, etc.]. The distribution arguments are defined in the PyMC documentation: https://pymc-devs.github.io/pymc/.

Storage_backend determines which format to store mcmc data, see self.avail_backends for a list of options.

fit(*q0=None*, *plot_residuals=False*, *plot_fit=False*, *opt_method='L-BFGS-B'*, *repeats=1*, *save_results=False*, *fname='opt_results.p'*)

Fits the deterministic model given in self.model to the data in self.data using ordinary least squares regression. Returns a parameter map containing the optimized model parameters and the associated sum of squared error.

The optional input, q0, should be an initial guess for the parameters in the form of a parameter map (dict).

The optional input repeats dictates the number of times the optimization process is repeated, each time using the previous result as the new q0.

generate_pymc_(*params*, *q0=None*)

Creates PyMC objects for each param in dictionary

NOTE: the second argument for normal distributions is VARIANCE

Prior option: An arbitrary prior distribution derived from a set of samples (e.g., a previous mcmc run) can be passed with the following syntax:

= {<name> : ['KDE', <pymc_database>, <param_names>]}

where <name> is the name of the distribution (e.g., 'prior' or 'joint_dist'), <pymc_database> is the pymc database containing the samples from which the prior distribution will be estimated, and <param_names> are the children parameter names corresponding to the dimension of the desired sample array. This method will use all samples of the Markov chain contained in <pymc_database> for all traces named in <param_names>. Gaussian kernel-density estimation is used to derive the joint parameter distribution, which is then treated as a prior in subsequent mcmc analyses using the current class instance. The parameters named in <param_names> will be traced as will the multivariate distribution named <name>.

generate_pymc_model(*q0=None*, *ssq0=None*, *std_dev0=None*, *fix_var=False*, *model_output_stored=False*)

PyMC stochastic model generator that uses the parameter dictionary, self. and optional inputs:

- **q0** [a dictionary of initial values corresponding to keys] in
- std_dev0 : an estimate of the initial standard deviation
- **ssq0** [the sum of squares error using q0 and self.data. Only] used if initial var, var0, is None.
- **fixed_var** [determines whether or not variance will be sampled] (i.e., fixed_var == False) or fixed.

pymcplot()

Generates a pymc plot for each parameter in self.. This plot includes a trace, histogram, and autocorrelation plot. For more control over the plots, see MCMCplots module. This is meant as a diagnostic tool only.

sample(*num_samples*, *burnin*, *step_method='adaptive'*, *interval=1000*, *delay=0*, *tune_throughout=False*, *scales=None*, *cov=None*, *thin=1*, *phi=None*, *verbose=0*)

Initiates MCMC sampling of posterior distribution using the model defined using the generate_pymc_model method. Sampling is conducted using the PyMC module. Parameters are as follows:

- num_samples : number of samples to draw (int)
- burnin : number of samples for burn-in (int)
- adaptive : toggles adaptive metropolis sampling (bool)

- **step_method** [step method for sampling; options are:] o adaptive - regular adaptive metropolis o DRAM - delayed rejection adaptive metropolis o metropolis - standard metropolis aglorithm
- **interval** [defines frequency of covariance updates (only] applicable to adaptive methods)
- **delay** [how long before first cov update occurs (only applicable] to adaptive methods)
- **tune_throughout** [True > tune proposal covariance even after] burnin, else only tune proposal covariance during burn in
- **scales** [scale factors for the diagonal of the multivariate] normal proposal distribution; must be dictionary with keys = .keys() and values = scale for that param.
- phi : cooling step; only used for SMC sampler

save_model (*fname='model.p'*)
Saves model in pickle file with name working_dir + fname.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX