Bayesian parameter estimation

General description : The algorithm is designed for the estimation of material parameters from given experimental data, the posterior probability distribution of each of the estimated material parameters and the joint probability distribution for combination of every two parameters using Bayes theorem. The posterior probability maps give us the credible interval for each parameter whereas the joint probability distribution of every two parameters give us the correlation between the two parameters.

Requirements : (1) Experimental data & (2) a Device model to generate training data for the neural network surrogate model, which eventually replaces the device model and speeds up the process of parameter estimation my many orders of magnitude.

Choice of the device model : The device model should be able to reproduce the experimental output as a function of material parameters or experimental conditions. The choice of device model depends on the parameters we want to estimate. Also, if there are multiple choices available, it makes sense to choose one that is fast as well as can be easily scripted to run in parallel.

Workflow :

1. Experimental data generation.
2. Identification of parameters that we want to estimate and then training data generation using the device model.
3. Training of the neural network surrogate model.
4. Finding a good starting point for the walkers in MCMC to start sampling the entire space.
5. Relocation of all the walkers to high probability region.
6. Start running the walkers to scan the entire parameter space and determine the posterior probability and the joint probability distribution using Bayes theorem.

Features :

1. Neural network surrogate model : Currently we are using a convolutional neural network to act as our surrogate model.

2. Automatic determination of a good starting point : In this step, given the experimental data, we determine a set of parameters that fits the data well enough. The point in the parameter space so determined then represents a high−probability starting point. To find the high−probability starting point we currently use CMA−ES (covariance matrix adaptation evolution strategy) algorithm from the pymoo (Multi objective optimization in Python) library.
   (https://pymoo.org/algorithms/soo/cmaes.html)
   Other algorithms can also be used, and I have tested the system with U−NSGA III, but for my current problem CMA−ES has given the best performance.
   However, other algorithms can be very easily incorporated into the code depending on the problem in hand.

3. Relocation of all the walkers to high probability region : Once a good starting point is determined, we can start running the MCMC. For this we need to give every walker a

unique starting point. Usually, it is done by adding random numbers to the starting point determined in the last step. However, when we are dealing with a high dimensional parameter space, adding random numbers often displaces the point from high probability to low probability space. This is due to the fact, that in higher dimensions, the probability landscape can often be very narrow. To avoid this issue, we run the MCMC algorithm for a bit and select the top 90% of walkers based on probability. K-means is there used with n-clusters and the nearest point to each cluster center is used with n being the number of walkers. This process is repeated several times until the maximum probability found remains stable from step to step. This ensures all the walkers start in a relatively high probability region.

4. MCMC : Once n–unique starting points have been determined, we start running MCMC, in which n walkers scan the entire parameter space and at each point they sample, we determine the posterior probability using bayes theorem. The convergence of MCMC is determined automatically using autocorrelation time analysis. We use the MCMC implementation of emcee library in this step (https://emcee.readthedocs.io/en/stable/).