SNPE Algorithm Write Ups

Throughout this – the example we are focussing on is Gaussian, so analytic posteriors that are easily tractable. The simulator takes the input theta (3D) and adds a linear shift of -1.0 and then adds a noise equal to 0.3 * standard normal distribution. Our observed value is (0,0,0). Therefore, our analytic marginal posteriors are Gaussians with mean 1.0 and variance 0.09, which we can plot. The prior is a box uniform between -2 and 2 in each dimension.

SNPE-B Algorithm

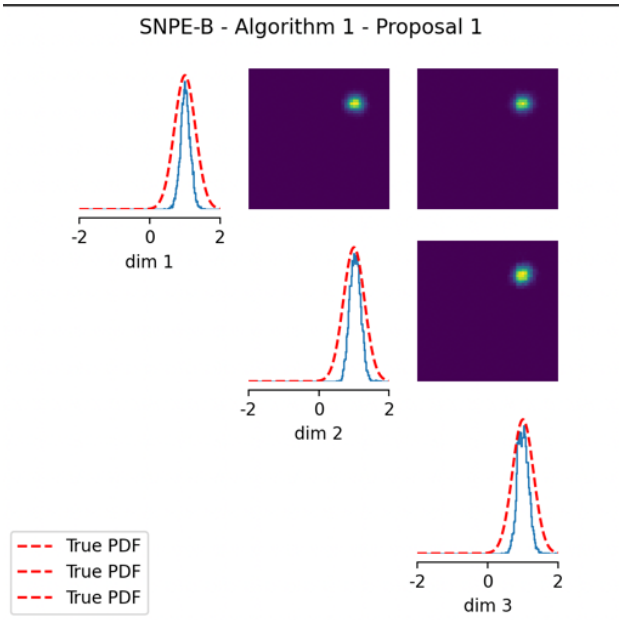
This is related to the notes you sent – there is a bit of confusion though, as outlined below.

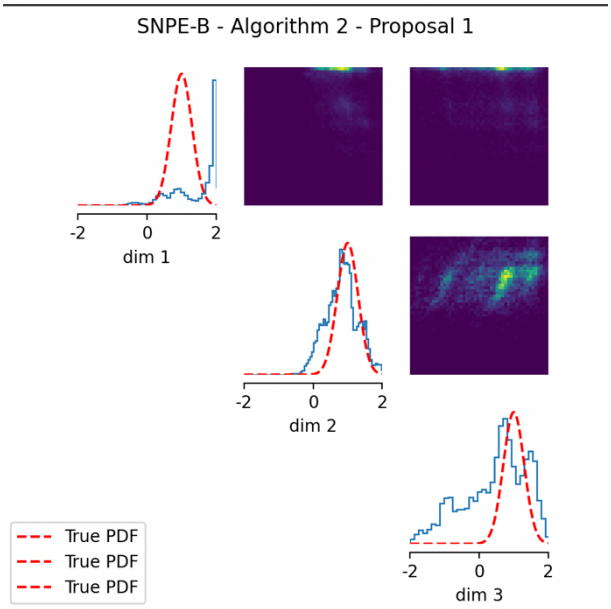
Algorithm 1: in round 1, proposal = prior. Train the density estimator on a modified loss function, given by loss_modified = - (log_prob + log_weights).mean(). This is made up of the usual loss function in NPE, which is -log_prob.mean(), and the extra importance weighted term, log_weights = prior_log_prob(theta) – proposal_log_prob(theta). For round 1 therefore, SNPE-B and NPE are the same. At the end of each round, the proposal is updated, you sample the thetas from the proposal and simulate from these proposal samples, and train with the modified loss function.

Algorithm 2: same as above, but the loss_modified = -exp(log_weights)*log_prob.mean(), so you don't add on the log importance weights but instead multiply on the importance weights. This is what the snpe_b.py code in the SBI library suggests doing (although this is technically "Not Yet Implemented") and references the following paper: https://arxiv.org/pdf/1711.01861 . The loss function linked in this paper has a couple of interesting features – firstly, there is a calibration kernel included which is not included in the SBI library, to remove simulated values far from the observation. Secondly, they say that to make the process more simulation efficient, they use all simulations, including from previous rounds, and using a stochastic variational inference loss function which looks a lot more involved.

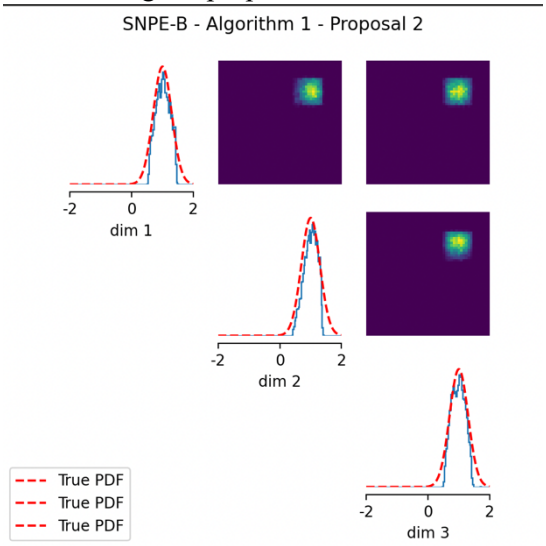
Proposal choice: Two options tried: **1**) the proposal is the posterior generated from the trained density estimator at the end of that round and the observation. **2)** based on the posterior generated from the trained DE and the observation, produce a truncated prior which is uniform in a smaller region of prior space (95% HPD).

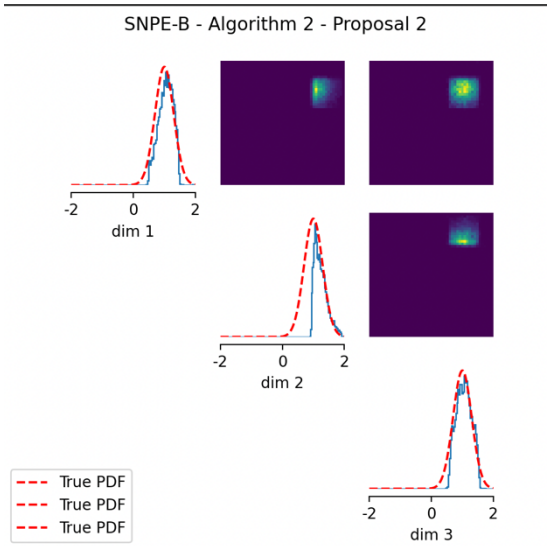
Posteriors generated: Here are the posteriors using 4 rounds of 10 epochs each, with the proposal set to choice 1, for algorithms 1 and 2:





And using the proposal choice 2:

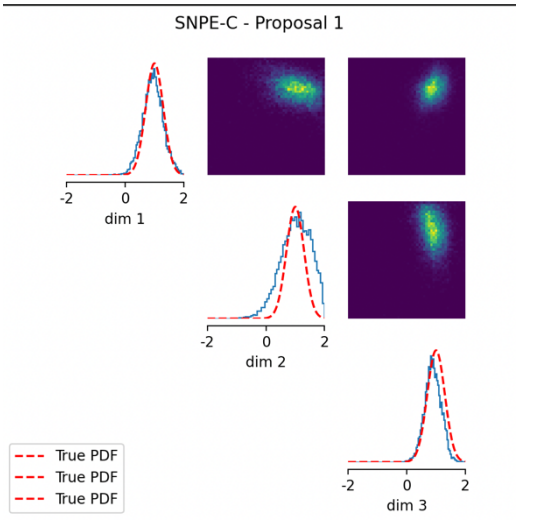


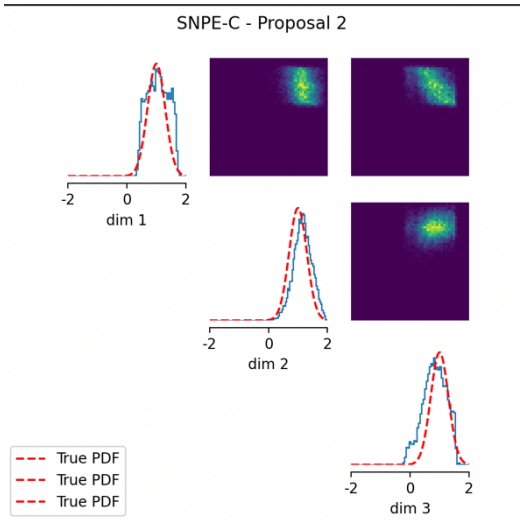


SNPE-C Algorithm

This is quite different to the one described above – the proposal modification to the loss is more subtle.

Algorithm: in round 1, proposal = prior. The modified loss function can take one of two forms, although I think the second is more common, and is what I've done here. In both cases, you calculate the **contrastive loss.** This is defined by setting a number of atoms, and then setting some sort of random choice to choose from the theta values that are not along the diagonal (check this – quite complex). Then you define an atomic_theta, which is the concatenation of the original theta and this contrastive_theta defined above. After some reshaping of the theta and the data, the log_prob_proposal_posterior is defined as follows:
unnormalized_log_prob = log_prob(atomic_theta, x) – prior.log_prob(atomic_theta)
log_prob_proposal_posterior is this normalized.
Then the loss is either: loss = -log_prob_proposal_posterior.mean() OR
- (log_prob_proposal_posterior + log_prob_non_atomic).mean()
Where log_prob_non_atomic = the non-proposal corrected log prob, i.e. standard NPE loss
We have used the second option. Then you replace the proposal and repeat rounds as before.
From my understanding, there is no explicit proposal dependence in the loss function. Instead, it seems the proposal merely influences the resampling of the data.
Posteriors generated: Here are the posteriors for proposal choices 1 and 2

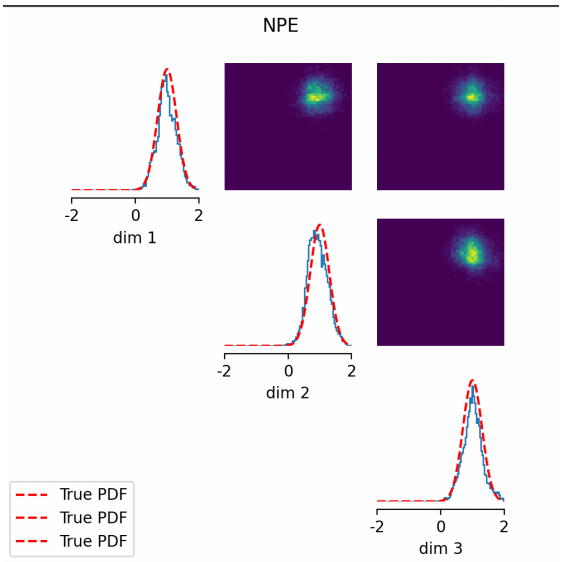




SNPE Algorithm Write Ups

NPE Algorithm:

This is the SNPE-B algorithm 1 with 1 round only – the importance weighting term adds on 0 to the loss. We run 1 round of this with 60 epochs to see what we get. The posteriors are as follows:



WandB loss curves: All available and labelled as NPE, SNPE-B Algorithm 1 Proposal 2, SNPE-C Proposal 1 etc. in the test_sequential_loops project.

Discussion:

- SNPE-B Algorithm 1 with proposal 1- overly confident posteriors – clearly proposal is too narrow to use
- SNPE-B Algorithm 2 with proposal 2 – poor inference, posteriors too wide
- SNPE-C with proposal 1 – seems quite good inference, more training probably needed, not sure if this is by increasing number of epochs per round or increasing number of rounds
- SNPE-C with proposal 2 – also seems good, more training probably needed, might also benefit from a different level of truncation – is 95% the right number?
- NPE – very accurate posteriors – ideally can recreate this with fewer simulations in the SNPE case
- SNPE-B points: I think algorithm 1 is the way to go unless we can recreate the more complete loss functions in algorithm 2 that's listed in the paper
- SNPE-C: The purpose of the contrastive atoms stuff isn't obvious to me, but it clearly works to some extent. The choice of number of atoms was not at all clear in the SBI library to me, have chosen 10 from a ChatGPT suggestion, but not clear if this is going to be a major factor in the quality of the inference
- SNPE-C: Is adding on the log_prob_non_atomic the right choice? In the SBI library it is given as a choice to use when the priors are bounded, I believe, so have used it here (and would be useful in PEREGRINE)?
- SNPE-B vs SNPE-C: because of this atoms stuff, the number of evaluations per epoch is much higher in SNPE-C so inference is much slower