Problem 1

(1)

By integrating the pdf, we have the CDF of standard Laplace distribution is

$$F(x) = \begin{cases} \frac{1}{2}e^{x} & x < 0 \\ 1 - \frac{1}{2}e^{-x} & x \geq 0 \end{cases}$$

So, to generate a standard Laplace random variable, first generate $x \sim U(0,1)$, then by solving $F(y) = x$, we have y being a sample from standard Laplace distribution. The correctness of this algorithm is shown in class by verifying $P(a \leq y \leq b) = F(b) - F(a)$.

Algorithm 1: get_laplace_sample

Result: sample from standard Laplace distribution

begin

  generate $x \sim U(0,1)$

  if $x \geq \frac{1}{2}$ **then**

    $res = -\log(2 - 2x)$

  else

    $res = \log(2x)$

  end

  return *res*

end

Below is a histograph of sampling results with groundtruth function.

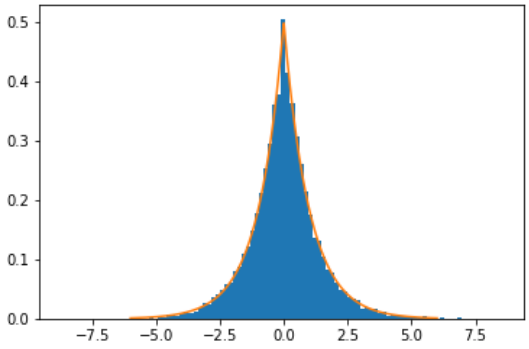

Figure 1: histograph of samples generated

(2)

To guarantee k times Laplace density can serve as a envelop function for standard Gaussian $N(0,1)$, we solve

$$\frac{1}{2}ke^{-|x|} \geq \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}, \quad \forall x$$

$$k \geq \sqrt{\frac{2e}{\pi}}$$

So the rejection sampling algorithm is shown below with $k = \sqrt{\frac{2e}{\pi}}$.

Problem 1 continued on next page...

Algorithm 2: sample_normal_from_laplace

begin
  laplace sample x_L = get_laplace_sample()
  accept probability $p_{acc} = e^{-\frac{1}{2}(|x|-1)^2}$
  decision $accept = Binomial(p_{acc})$
  if $accept == 1$ **then**
    return x_L
  else
    run again
  end
end

And sampling result is shown below.

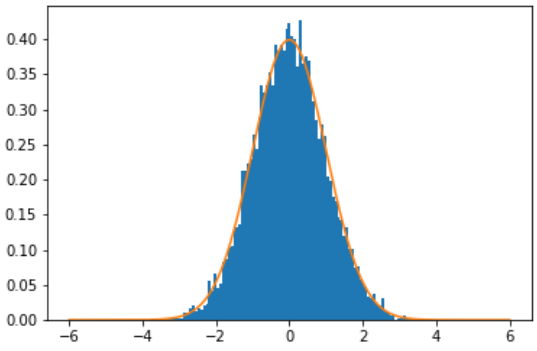

Figure 2: histograph of samples generated

(3)
No. Similar to the choice of k in **(2)**, we have

$$\frac{1}{2}e^{-|x|} \leq \frac{k}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}, \quad \forall x$$

No $k \in R$ satisfy the above equation as $x \to \infty$, so we cannot simulate Laplace RV using $N(0,1)$ as envelop function.

Problem 2

(1)
First, we have the probability of (μ, σ) given the observed data (up to a normalsizing constant)

$$\begin{aligned} p(\mu, \sigma^2|X) &\propto p(\mu)p(\sigma^2)p(X|\mu,\sigma^2) \\ &\propto e^{-\frac{(\mu-\mu_0)^2}{2\tau_0^2}} \frac{e^{-\frac{\nu_0\sigma_0^2}{2\sigma^2}}}{\sigma^{2+\nu_0}} \prod \frac{1}{\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \end{aligned}$$

To derive the conditional probability $p(\mu|\sigma^2, X)$, we can a) simply ignore all terms only related to σ, since they will be integrated out and hence constant, and b) for all terms related to μ, treat the σ as a fixed

constant. We then have

$$\begin{aligned} p(\mu|\sigma^2, X) &\propto exp(-\frac{(\mu-\mu_0)^2}{2\tau_0^2} - \sum \frac{(x_i-\mu)^2}{2\sigma^2}) \\ &\propto exp(-(\frac{1}{2\tau_0^2} + \frac{n}{2\sigma_2})\mu^2 + (\frac{\mu_0}{\tau_0^2} + \frac{\sum x_i}{\sigma^2})\mu) \\ &\propto exp(-\frac{(\mu - \frac{\frac{\mu_0}{\tau_0^2} + \frac{\sum x_i}{\sigma^2}}{\frac{1}{\tau_0^2} + \frac{n}{\sigma_2}})^2}{2\frac{1}{\frac{1}{\tau_0^2} + \frac{n}{\sigma_2}}}). \end{aligned}$$

This implies $\mu \sim N(\frac{\frac{\mu_0}{\tau_0^2} + \frac{\sum x_i}{\sigma^2}}{\frac{1}{\tau_0^2} + \frac{n}{\sigma_2}}, \frac{1}{\frac{1}{\tau_0^2} + \frac{n}{\sigma_2}})$. Similarly,

$$p(\sigma^2|\mu, X) \propto \frac{1}{\sigma^{2+\nu_0+n}} e^{-\frac{\nu_0\sigma_0^2 + \sum(x_i-\mu)^2}{2\sigma^2}},$$

which means $\sigma^2 \sim \text{Inv-}\chi^2(\nu_0 + n, \frac{\nu_0\sigma_0^2 + \sum(x_i-\mu)^2}{\nu_0+n})$.

(2)

Algorithm 3: gibbs_sampler

Result: sample from gibbs sampling
begin
  generate $\mu \sim N(0, 1)$
  generate $\sigma^2 \sim \text{Inv-}\chi^2(1, 1)$
  while $\#\ of\ samples \leq 1000$ **do**
    $\mu \sim p(\mu|\sigma^2, X)$
    $\sigma^2 \sim p(\sigma^2|\mu, X)$
  end
end

Trace plot of μ, σ is shown below.(Note: not σ^2)

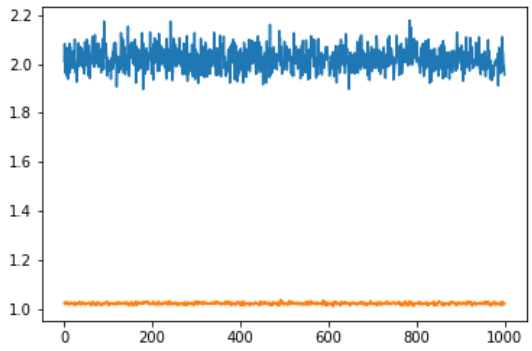

Figure 3: trace plot of samples generated

(3)

Algorithm 4: metropolis_sampler

Result: sample from metropolis sampling

begin

generate $\mu \sim N(0,1)$

generate $\sigma^2 \sim$ Inv-$\chi^2(1,1)$

while # *of samples* ≤ 1000 **do**

$\mu' \sim N(\mu|step\_mu)$

$\sigma'^2 \sim exp(U(-step\_sigma, step\_sigma))\sigma^2$

accept the new μ', σ' with probability

$\min(1, exp(\text{log\_likelihood}(\mu', \sigma'^2) - \text{log\_likelihood}(\mu, \sigma^2)))$

end

end

Trace plot of μ, σ is shown below with step_mu and step_sigma being 0.04.(Note: not σ^2)

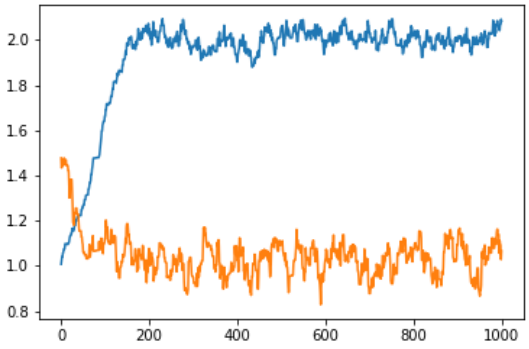

Figure 4: trace plot of samples generated

(4)

Trace plots and ACF plots are shown below, with $i \geq 200$ being burn-in phase, the first 200 sample are therefore discarded for ACF.

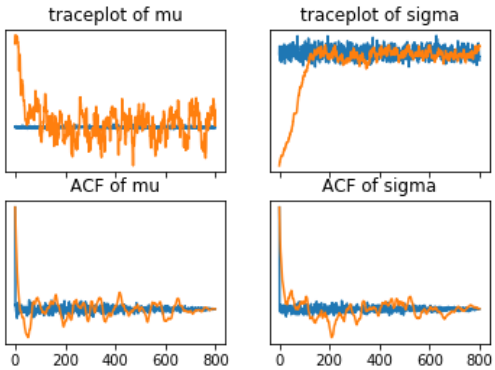

Figure 5: trace plot of samples generated, above: 1k samples, below: 800

Gibbs sampler is better in this case, which has more stable and less correlated samples, and burns in fast. That is because Gibbs sampler makes use of the conditional distribution in this problem.

Problem 3

(1)
The scatter plot of HMC samples are shown below.

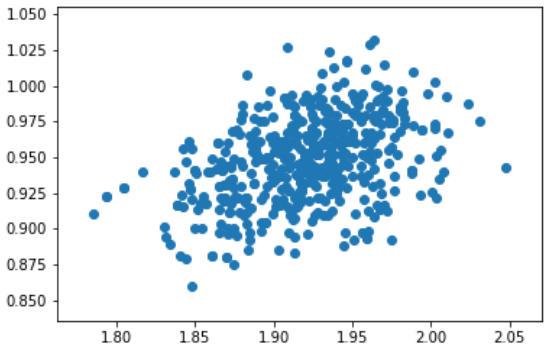

Figure 6: scatter plot of samples generated by HMC

Below are figures of ACFs of samples by a) HMC with fixed L, b) HMC with random L, as L increase from 10 to 90. The left 3 * 3 grid are results from HMC with fixed L, while the right 3 * 3 from HMC with random L. We see samples are more stable and robust to change of L when L are sampled. That is because when L are sampled, HMC are able to choose a diverse L, which will 1) enable HMC to sample from local points **and** 2) enable HMC to tarvel far away out of possible local maxima, whereas a fixed L will force HMC to either 1) travel far away **or** 2) stay in local modes.

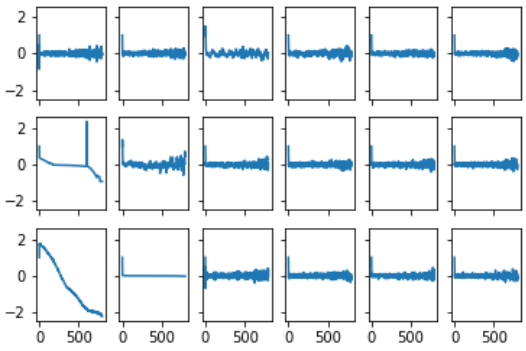

Figure 7: ACF of fixed L(left), and random L(right)

(2)
We evaluate each algorithm by applying a sliding window of 1k samples from each algorithm and calculate the KL divergence to GT 1k samples from HMC(down sampled from 50k samples). The above plot are smoothed by a Gaussian filter(with sigma 4). As for implementation and hyperparameters, we applied lr decay for SGLD, and not for SGHMC and SGNHT. SGHMC and SGNHT are runned with $\epsilon = 0.001, C = 10$, which yields best results in SGNHT's original paper and turns out to be the best in my experiments. We

show the results of different SGHMC methods in terms of KL divergence to GT(by a long HMC) below.

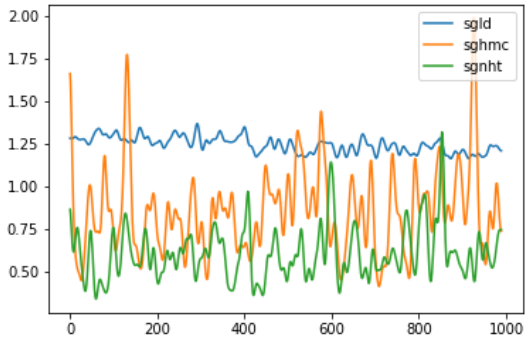

Figure 8: KL with batch_size=32

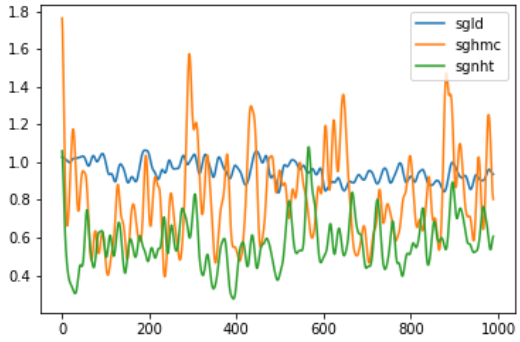

Figure 9: KL with batch_size=64

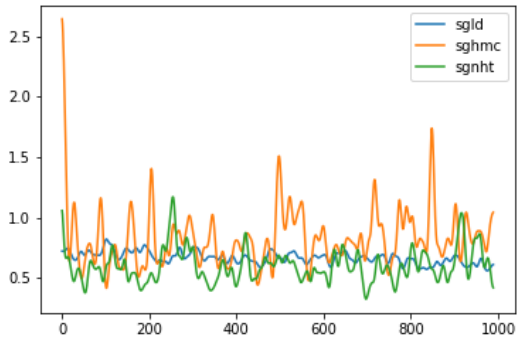

Figure 10: KL with batch_size=128

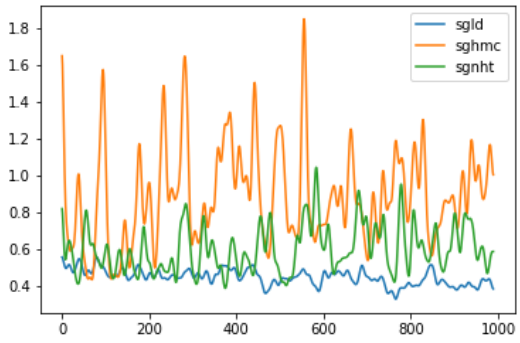

Figure 11: KL with batch_size=256

We can conclude that

- as for performance, SGLD < SGHMC < SGNHT
- for convergence rate and average KL after convergence, SGLD < SGHMC < SGNHT(refer to the plot below for more consistant comparation with lr being the same for all methods)
- SGLD seems to be stuck somewhere, while SGHMC and SGNHT are generally better(I chose a small learning rate for SGHMC and SGNHT so the var. seems quite high)
- SGLD is more sensitive to batch size and learning rate, we also show a plot of three algorithms with the same bs and lr.

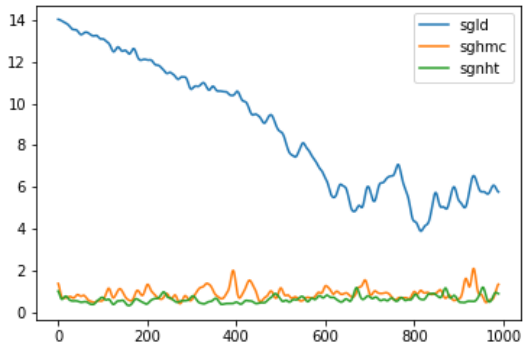

Figure 12: KL of different SGHMC methods with the same lr