STA 250 HOMEWORK 4

Yichuan Wang

Problem 1

In the first problem of this homework, I implemented the method proposed by Robert (2009) in CUDA code in order to obtain samples from a truncated normal distribution with given parameters. The distribution can be represented as

$$TN(\mu, \sigma^2; a, b)$$

where μ and σ^2 are the mean and variance of the original, i.e. untruncated, normal distribution and a, b represent the lower and upper bounds of the truncation interval with $a \geq -\infty, b \leq \infty$. Provided in the lecture notes, the expected values for a random variable $Z \sim TN(\mu, \sigma^2; a, b)$ is

$$E(Z) = \begin{cases} \mu - \sigma \frac{\phi(\frac{b-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma})} & \text{if } a = -\infty, b < \infty \\ \mu + \sigma \frac{\phi(\frac{a-\mu}{\sigma})}{1-\Phi(\frac{a-\mu}{\sigma})} & \text{if } a > -\infty, b = \infty \\ \mu + \sigma \frac{\phi(\frac{a-\mu}{\sigma}) - \phi(\frac{b-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})} & \text{if } a > -\infty, b < \infty \end{cases}$$

where ϕ, Φ are p.d.f and c.d.f of the standard normal distribution respectively. As directed in the paper by Robert (2009), when the truncation region is one-sided, for example $TN(0, 1; \mu^-, \infty)$, the algorithm for sampling from such truncated normal goes as follows:

(1) Generate a random sample $z = \mu^- + Expo(\alpha)$, where $Expo(\alpha)$ stands for a random number generated from the exponential distribution with parameter α.

(2) Compute the value $\psi(z)$ where

$$\psi(z) = \begin{cases} \exp(-\frac{(\alpha-z)^2}{2}) & \text{if } \mu^- < \alpha \\ \exp(-\frac{(\mu^- -\alpha)^2}{2}) \exp(\frac{(\alpha-z)^2}{2}) & \text{if } \mu^- \geq \alpha \end{cases}.$$

(3) Sample u from $U[0,1]$, if $u < \psi(z)$ then we accept z as our sample for this iteration; otherwise go back to step (1).

When the truncation region is one-sided but with an upper truncation, for example $TN(0,1;-\infty,\mu^+)$, we may simple reverse the sign of μ^+ and sample from lower truncation case then reverse the sign back. Also the optimal choice of parameter α for the exponential distribution is $\alpha_* = \frac{\mu^- + \sqrt{(\mu^-)^2+4}}{2}$.
When we have a two-sided truncation, e.g. $TN(0,1,;\mu^-,\mu^+)$, the algorithm is:

(1) Sample z from $U[\mu^-,\mu^+]$.

(2) Compute $\psi(z) = \begin{cases} \exp(-\frac{z^2}{2}) & \text{if } 0 \in [\mu^-,\mu^+] \\ \exp(-\frac{(\mu^-)^2-z^2}{2}) & \text{if } \mu^- > 0 \\ \exp(-\frac{(\mu^+)^2-z^2}{2}) & \text{if } \mu^+ < 0 \end{cases}$

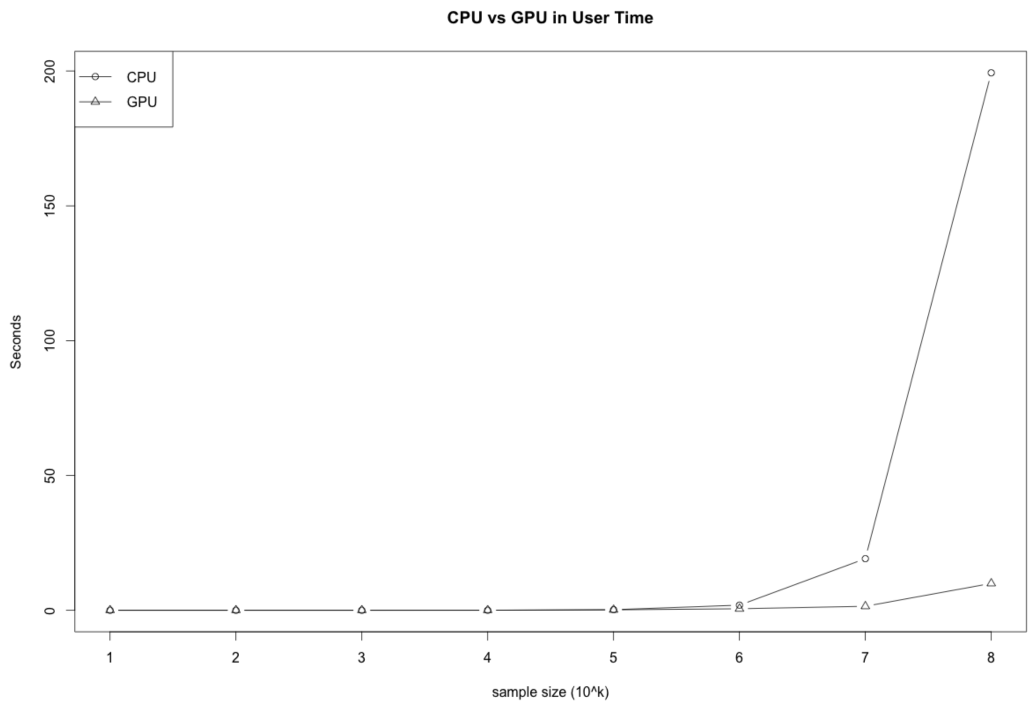
(3) Sample u from $U[0,1]$, if $u < \psi(z)$, we accept z; otherwise, go back to step (1).

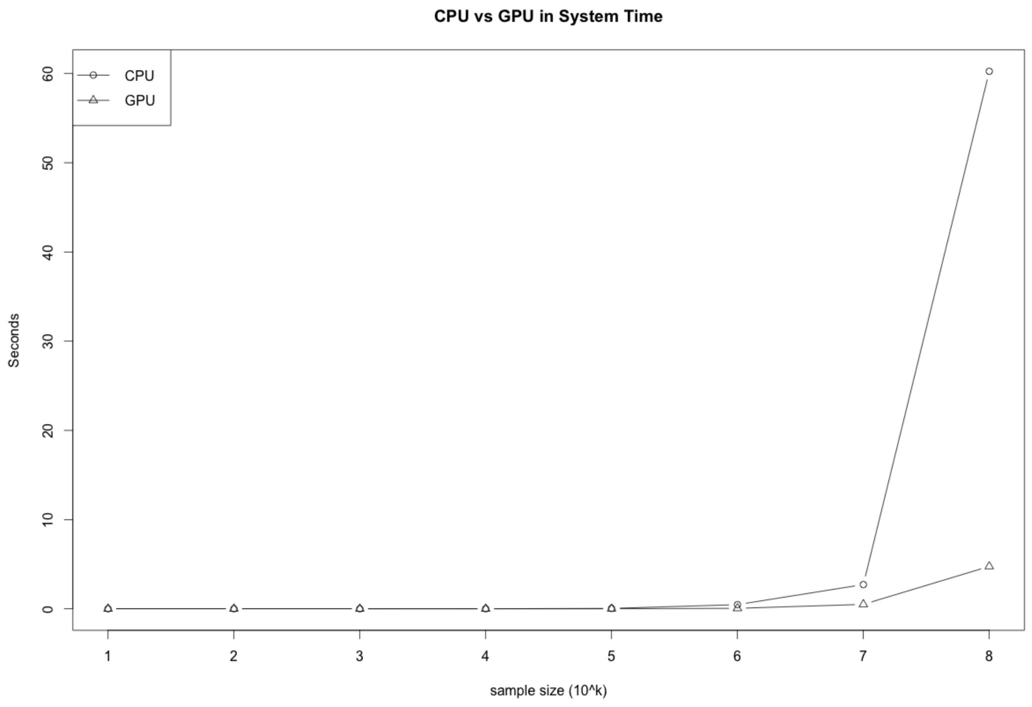
For any truncated normal distribution where $\mu \neq 0$ and/or $\sigma \neq 1$, we may simple use location-scale transformation to obtain properly transformed truncation region then transform the samples back to corresponding ones in the desired distribution.
After executing the code on AWS to generate 10000 samples from distribution $TN(2,1;0,1.5)$, the expected value was calculated to be 0.957, and the mean of samples were 0.849. The two values are not very close but not too far away. It could be due to some coding error when implementing the algorithm from Robert (2009).
Another trial was executed in R with the function rtnorm() from "msm" package to generate 10000 samples from the same truncated normal distribution. This time the mean of 10000 random samples turned out to be 0.954, which is quite close to the expected value.
Then both the CUDA and R code were put to test to generate 10^k samples, $k = 1,\cdots,8$, still from the distribution $TN(2,1;0,1.5)$. The mean values for samples generated by GPU and CPU are

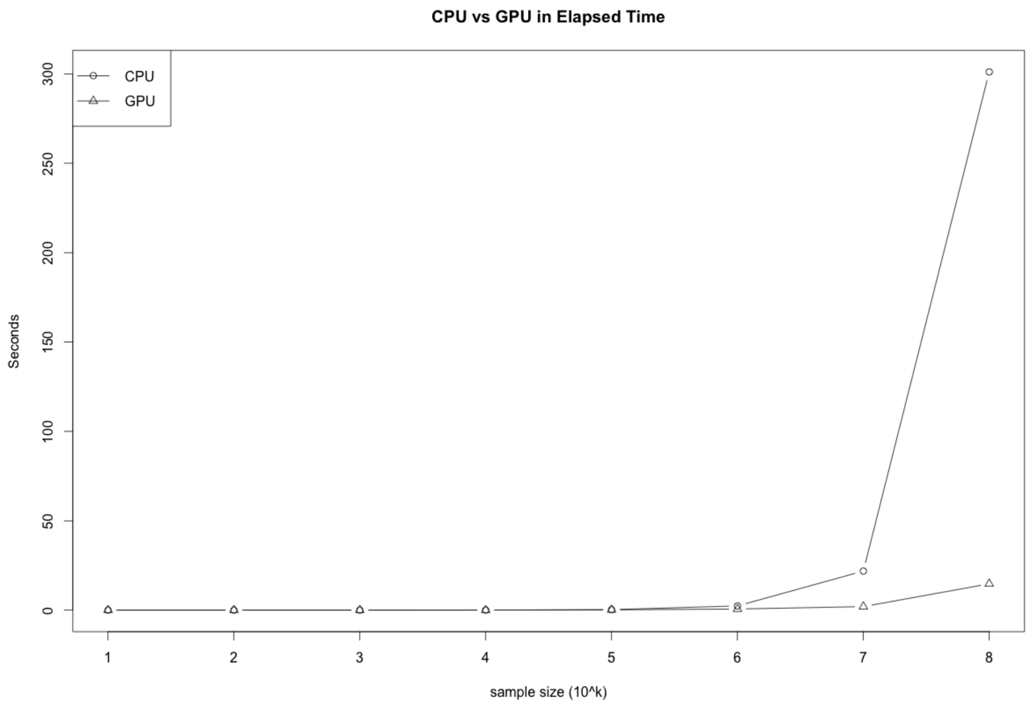
$k =$	1	2	3	4	5	6	7	8
GPU	0.871	0.812	0.843	0.849	0.849	0.849	0.849	0.849
CPU	1.343	0.946	0.950	0.954	0.957	0.957	0.957	0.957

Both CUDA and R codes returned "converged" mean values when sample size became large. However, the R code gave closer estimate of the expected value for the truncated normal distribution.

When looking at the cost/time of completing the tasks, GPU showed significant gain over CPU in terms of requiring less time to complete tasks when the sample size is large. The following are three figures, each for "User Time", "System Time" and "Elapsed Time":







Apparently when sample sizes are small ($k = 1, \cdots, 6$), GPU did not offer

much difference from CPU; however, when sample size got large ($k = 7, 8$), the time needed for CPU to complete each task grew exponentially, whereas GPU only required moderately more time than smaller sample sizes. Also most of the time increased for GPU was the "User Time", namely time needed transferring data between host and device.
For verification purposes, both CUDA and R codes were tested with three different truncated normal distributions: $TN(2,1;-\infty,0), TN(2,1;0,\infty)$, $TN(0,1;-\infty,-10)$. 10000 samples were generated with GPU and CPU respectively, and the means are listed below together with expected values:

Distribution	$E(Z)$	GPU	CPU
$TN(2,1;-\infty,0)$	-0.373	-0.374	-0.374
$TN(2,1;0,\infty)$	2.055	2.057	2.057
$TN(0,1;-\infty,-10)$	-10.098	-10.098	-10.098

The results in the table above showed that both my GPU and CPU code works for one-sided truncated normal distributions as well as in the tail region.

Problem 2

In this problem, we implemented Gibbs sampler algorithm for Probit MCMC, whose model is given by

$$\begin{aligned} Y_i|Z_i &\sim \mathbf{1}_{\{Z_i>0\}} \\ Z_i|\beta &\sim N(x_i^T\beta, 1) \\ \beta &\sim N(\beta_0, \Sigma_0) \end{aligned}$$