Probing The Slice Sampling Algorithm

Henry Agbi-Kaiser

2024-12-11

1 Introduction

In Bayesian statistics, MCMC techniques such as Gibbs sampling and the Metropolis algorithm are widely used to sample from complicated, high-dimensional distributions. However, their implementation is rather involved because one often needs to come up with proposal distributions for Metropolis or standard form for conditional distributions when using Gibbs. Neal (2003) introduced slice sampling, an adaptive method that requires less tuning and hence more suitable for automatic implementation, possibly for routine use in Bayesian analysis.

In slice sampling, the vertical and horizontal dimensions under a distribution's density plot are alternately sampled to create a "slice" from which samples are drawn uniformly. This is particularly useful for distributions in which there are large changes of scale, since slice sampling adapts to such changes dynamically. Slice sampling can also avoid the intrusive random walks using techniques such as overrelaxation and adaptive updates, which may make it more efficient for certain kinds of problem than other MCMC methods.

The objective of the project is to carry out in-depth research on slice sampling as an independent technique within the context of Bayesian statistics. This project is meant to focus purely on mechanics, implementation, and performance of slice sampling without any direct comparison to other MCMC methods. In that regard, this project gives some insight into single-variable slice sampling efficiency and just how versatile it is in accommodating all kinds of distributions.

2 Methodology

2.1 Overview

This project follows the framework specified by Neal (2003) in his seminal paper on slice sampling. The focus is on implementing single-variable slice sampling, evaluating its performance across diverse distributions, and analyzing its accuracy and efficiency using well-defined metrics. The methodology includes algorithm implementation, simulation of samples from selected target distributions, and performance evaluation through quantitative and visual diagnostics.

2.2 Slice Sampling Algorithm

The single-variable slice sampling technique generates samples by iteratively sampling from the a target probability density function (PDF) through an iterative process of defining a "slice" and choosing samples uniformly within it. The procedure is as follows:

1. **Defining the Slice**
   An auxiliary variable y is sampled uniformly from the interval $(0, f(x_0))$, where $f(x)$ is the target PDF and x_0 is the current sample. This value y defines a horizontal slice $S = \{x : f(x) \geq y\}$ The slice S represents the region of the PDF such that $f(x) \geq y$, and x_0 is always in S.
2. **Finding the Slice Interval**
   The next step is to find an interval $I = (L, R)$ around x_0 that contains all or most of the slice S. This is done using the doubling procedure in Figure 1 , which starts with a small interval around x_0 and iteratively doubles its size until it covers the region where $f(x) \geq y$. This method ensures efficient exploration of the slice while minimizing unnecessary computations in subsequent iterations.
3. **Sampling a New Point**
   A new point x_1 is sampled uniformly from the interval I defined in the previous step. If x_1 lies outside the slice S, that is $f(x_1) < y$, the interval I is reduced iteratively using the shrinkage procedure in Figure 2 to ensure that the new point remains within the slice. This process guarantees that the sample respects the condition $f(x) \geq y$ maintaining the validity of the slice sampling algorithm. Since the doubling procedure was used, the point must pass the acceptance test in Figure 3.
4. **Repeating the Process**
   Steps 1 through 3 are repeated to generate a sequence of samples. Each new sample x_1 becomes the starting point x_0 for the next iteration.

The algorithm was implemented in base R (R Core Team (2024)) with Rstudio as the development environment for R (Posit team (2024)). The complete code for the algorithm is provided in Appendix 6.1.

Input:

f	$=$	function proportional to the density
x_0	$=$	the current point
y	$=$	the vertical level defining the slice
w	$=$	estimate of the typical size of a slice
p	$=$	integer limiting the size of a slice to $2^p w$

Output: $(L, R) =$ the interval found

```
$U \sim \text{Uniform}(0, 1)$
$L \leftarrow x_0 - wU$
$R \leftarrow L + w$
$K \leftarrow p$
repeat while $K > 0$
        and $\{y < f(L)$ or $y < f(R)\}$:
    $V \sim \text{Uniform}(0, 1)$
    if $V < 1/2$ then $L \leftarrow L - (R - L)$
                 else $R \leftarrow R + (R - L)$
    $K \leftarrow K - 1$
```

Figure 1: The doubling procedure for expanding an interval around the current point x_0. Adapted from Neal (2003).

Input:

f	$=$	function proportional to the density
x_0	$=$	the current point
y	$=$	the vertical level defining the slice
(L, R)	$=$	the interval to sample from

Output: $x_1 =$ the new point

```
$\bar{L} \leftarrow L, \quad \bar{R} \leftarrow R$

Repeat:
    $U \sim \text{Uniform}(0, 1)$
    $x_1 \leftarrow \bar{L} + U \cdot (\bar{R} - \bar{L})$
    if $y < f(x_1)$ and Accept($x_1$) then exit loop
    if $x_1 < x_0$ then $\bar{L} \leftarrow x_1$
              else $\bar{R} \leftarrow x_1$
```

Figure 2: The "shrinkage" procedure used to sample a new point x_1 from the interval $I = (L, R)$. Adapted from Neal (2003).

Input:

f	$=$	function proportional to the density
x_0	$=$	the current point
x_1	$=$	the possible next point
y	$=$	the vertical level defining the slice
w	$=$	estimate of the typical size of a slice
(L, R)	$=$	the interval found by the doubling procedure, using w

Output: whether or not x_1 is acceptable as the next state

```
$\hat{L} \leftarrow L, \quad \hat{R} \leftarrow R$
$D \leftarrow$ false
repeat while $\hat{R} - \hat{L} > 1.1 \cdot w$:
    $M \leftarrow \frac{\hat{L} + \hat{R}}{2}$
    if $\{x_0 < M$ and $x_1 \geq M\}$ or
       $\{x_0 \geq M$ and $x_1 < M\}$
        $D \leftarrow$ true
    if $x_1 < M$ then $\hat{R} \leftarrow x_1$
               else $\hat{L} \leftarrow M$
    if D and $y \geq f(\hat{L})$ and $y \geq f(\hat{R})$ then
        the new point is not acceptable

The new point is acceptable if it is
not rejected in the loop above
```

Figure 3: The test for determining whether a proposed point x_1 is acceptable within the slice $S \cap I$. The "while" condition, with a 1.1 multiplier, guards against potential rounding errors, and the variable D tracks if the intervals differ from those leading to the current point. Adapted from Neal (2003).

2.3 Target Distributions and Performance Evaluation

To evaluate the single-variable slice sampling algorithm, four target distributions of varying complexity were selected, as shown in Table 1.

Table 1: Target distributions used for evaluating the slice sampling algorithm, including their formulas, support domains, and parameters

Name	Formula	Support/Domain	Parameter
Beta	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$	$0 \le x \le 1$	$\alpha = 2, \beta = 6$
Inverse Gaussian (Wald)	$\sqrt{\frac{\lambda}{2\pi x^3}}\exp\left\{-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right\}$	$0 < x < \infty$	$\mu = 3, \lambda = 2$
Sinusoidal Exponential Decay(Sinu-Exp)	$\lvert\sin(x)\rvert \exp\{-x/4\}$	$0 < x < \infty$	
Bimodal Gaussian Mixture(Bimodal-GM)	$p \cdot \frac{1}{\sqrt{2\pi\sigma_1^2}}exp\left\{-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right\} + (1-p)\cdot\frac{1}{\sqrt{2\pi\sigma_2^2}}exp\left\{-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right\}$	$-\infty < x < \infty$	$p = 0.6,$ $\mu_1 = 0, \mu_2 = 5,$ $\sigma_1 = 1, \sigma_2 = 1.5$

These distributions were chosen to represent a range of challenges for the algorithm, including bounded supports, heavy tails, oscillatory patterns, and multimodality. Samples were simulated from each distribution at three different sample sizes ($n = 50, 500, 5000$) to assess the algorithm's performance under different scenarios.

To measure the accuracy and efficiency of the algorithm, the following performance metrics were employed:

1. **L2 Distance**
   The L2 distance quantifies the difference between the empirical and theoretical cumulative distribution functions (CDFs). It is calculated as:
   $$L_2 = \sqrt{\int_{-\infty}^{-\infty} (F(x) - F_n(x))^2 dx},$$
   where $F(x)$ and $F_n(x)$ are the empirical and theoretical CDFs, respectively. Smaller L2 distances indicate better alignment with the target distribution and provide a quantitative measure of accuracy.

2. **Empirical CDF and PDF Comparisons**
   Visual comparisons of the empirical and theoretical CDFs and probability density functions (PDFs) were performed. These plots illustrate how well the samples represent the underlying distribution and highlight deviations in regions such as the tails, peaks, or areas with rapid variation.

3. **Autocorrelation Function (ACF)**
   The ACF measures the dependency between successive samples in the Markov chain. At lag k, the autocorrelation is defined as:
   $$\rho_k = \frac{Cov(x_t, x_t + k)}{Var(x_t)},$$
   where ρ_k reflects the correlation between a sample and another k steps later. Faster decay in ρ_k values indicates efficient mixing, reducing dependencies and improving exploration of the target distribution.

4. **Effective Sample Size (ESS)**
   The ESS measures the number of independent samples effectively generated, accounting for autocorrelation. It is computed as:
   $$ESS = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho_k},$$
   where n is the total number of samples, and ρ_k is the autocorrelation at lag k. A higher ESS indicates that the algorithm efficiently produces independent samples, while a lower ESS suggests inefficiency due to high autocorrelation.

By combining these metrics, the algorithm's performance can be evaluated in terms of both accuracy (L2 distance and visual comparisons) and efficiency (ACF and ESS). This comprehensive assessment ensures that the slice sampling algorithm is robust across different distributions and sample sizes. The R code for computing the performance metrics is provided in Appendix 6.2, while the sampling procedures for each target distribution are detailed in Appendix 6.3.

3 Results

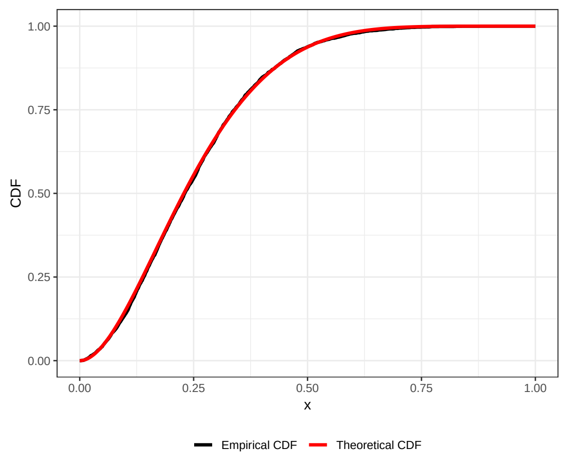
3.1 Accuracy Evaluation

The accuracy of the slice sampling algorithm was assessed using L2 distance, together with comparisons of the cumulative distribution function and probability density function. The L2 distance values for each distribution for every sample size are shown in Table 2. The results show that the L2 distance consistently decreases as the sample size increases, indicating that the slice sampling algorithm becomes more accurate as the sample size is increased. For example, the L2 distance for the Beta(2,6) distribution reduced from 0.001531 at $n = 50$ to 0.000010 at $n = 5,000$. A similar pattern was found in Wald(3,2), Sinu-Exp, and Bimodal-GM distributions. Such a reduction points to an improved ability of the algorithm in approximating the theoretical distribution with increasing sample size.

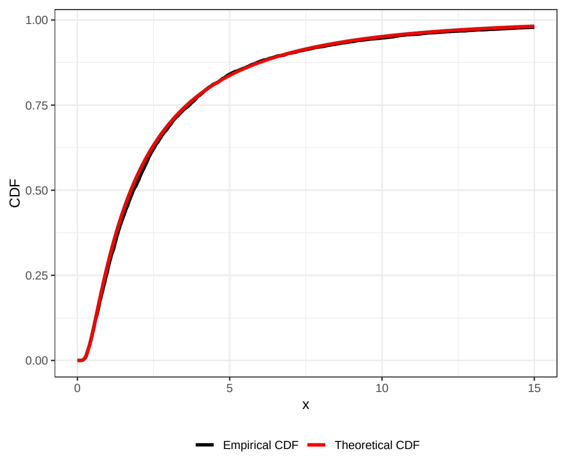
Table 2: L2 Distance Results

Sample Size	Distribution			
	Beta(2,6)	Wald(3,2)	Sinu-Exp	Bimodal-GM
50	0.001531	0.029834	0.037431	0.012067
500	0.000101	0.001544	0.008539	0.001707
5000	0.000010	0.000767	0.000747	0.000164

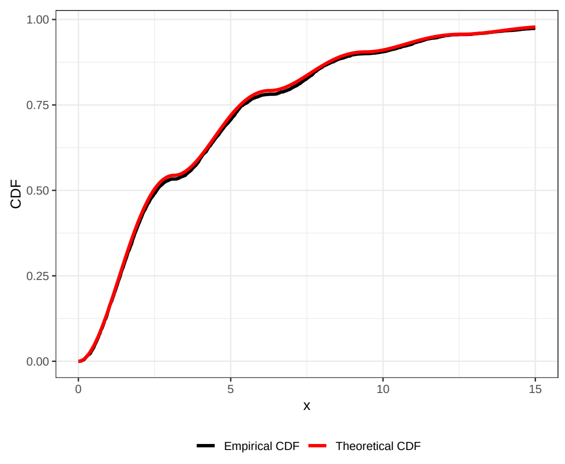
Figure 4 shows the theoretical and empirical CDFs for each distribution at $n = 5,000$. The empirical CDFs for Beta(2,6) and Bimodal-GM are very close to the theoretical CDFs, showing that the slice sampling algorithm works very well for these distributions, this is supported by their low L2 distances. For Wald(3,2), some minor deviations can be observed, reflecting its relatively larger L2 distance. Similarly, the Sinu-Exp distribution show slight deviations in oscillatory regions. This reflects the challenges of capturing this complex structure. These results evidence the algorithm's effectiveness as well as its limitations concerning distributions with heavy tails and rapid variations.



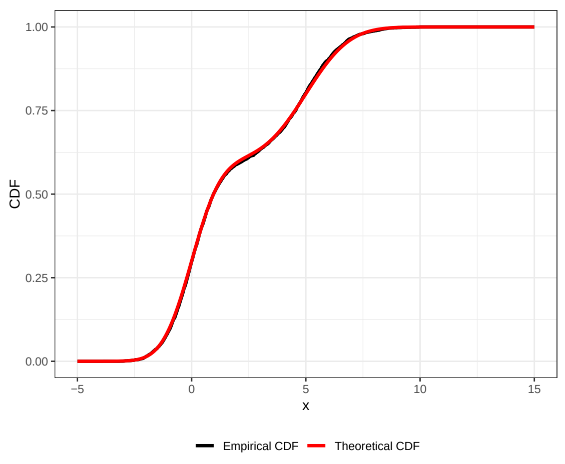
(a) Beta(2,6)



(b) Wald(3,2)



(c) Sinu-Exp



(d) Bimodal-GM

Figure 4: Empirical CDFs compared to the theoretical CDFs for the Beta(2,6), Wald(3,2), Sinu-Exp, and Bimodal-GM distributions at $n = 5000$. The close alignment of the empirical and theoretical curves demonstrates the accuracy of the slice sampling algorithm across diverse distributions.

The Probability Density Function comparisons for the Beta(2,6), Wald(3,2), Sinu-Exp, and Bimodal-GM distributions in Figure ref(fig:pdf) display the generally good performance of the slice sampling algorithm. For the Beta(2,6) distribution, the sampled density closely matches the theoretical density, with a slight underestimation in the peak. The Wald(3,2) distribution shows reasonable agreement, but the sampled density underestimates the peak and slightly overestimates as it decays. In the case of the Sinu-Exp distribution, the general oscillatory structure is captured, although some smoothing in regions of rapid oscillations can be seen. Lastly, for the Bimodal-GM distribution, both modes and their relative heights are effectively captured;

there is a slight deviation in the first mode at 0. These results point out the algorithm's adaptability for diverse distributions while underlining small inaccuracies in the most complex regions.

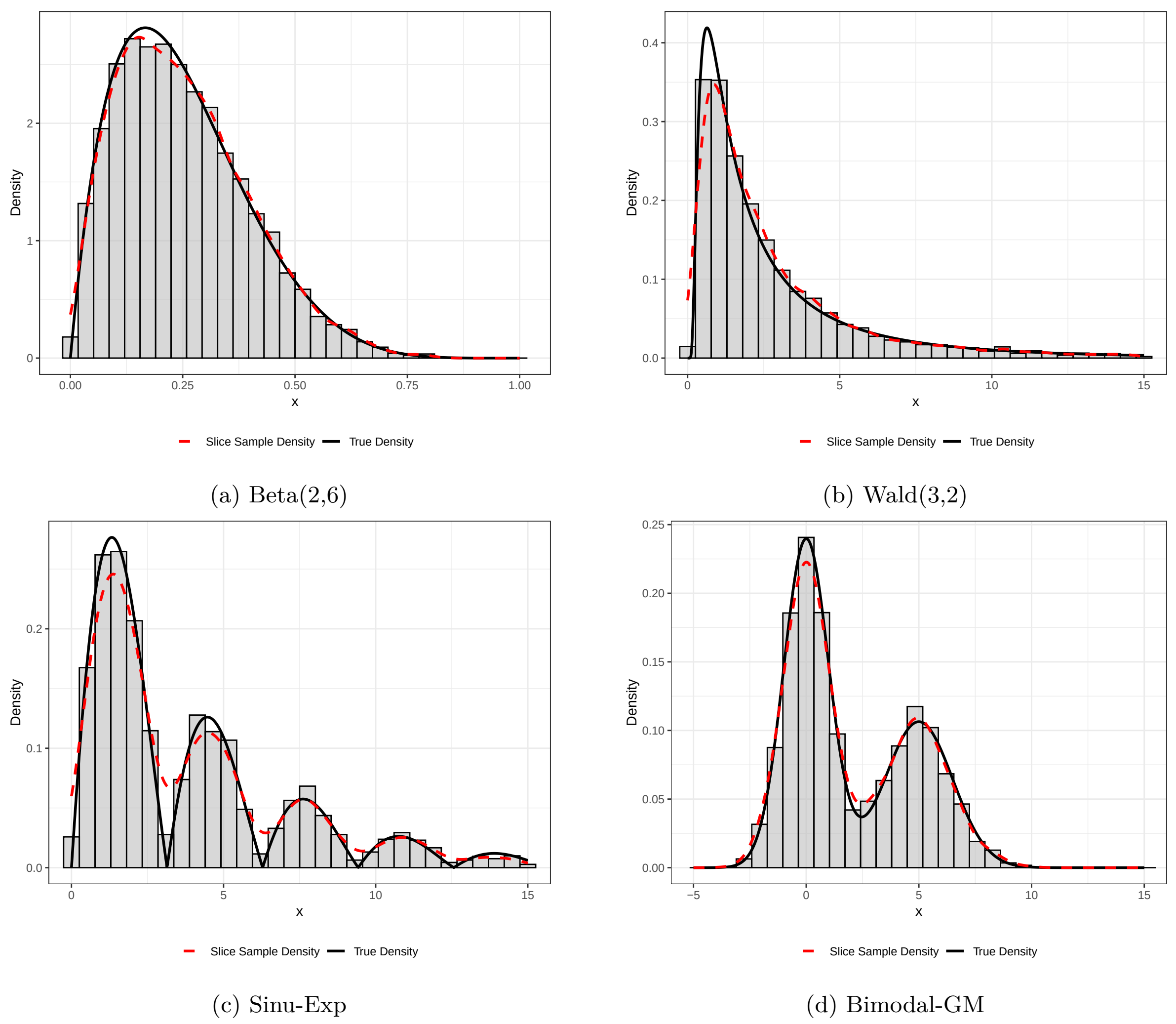


Figure 5: PDFs comparing the slice-sampled densities (red dashed lines) to the theoretical densities (black solid lines) for the Beta(2,6), Wald(3,2), Sinu-Exp, and Bimodal-GM distributions. The plots highlight the accuracy of the slice sampling algorithm, with minor discrepancies observed in regions of higher complexity, such as oscillatory structures.

In addition to L2 distance and visual comparisons of CDFs and PDFs, the accuracy of the slice sampling algorithm was further checked by comparing empirical and theoretical summary statistics for each distribution, as presented in Table 3.

In the case of the Beta(2,6) distribution, the empirical mean (0.2511) and standard deviation (0.1453) are almost identical to the theoretical values (0.2500 and 0.1443), with a very low MC error of 0.0024, indicating high precision and reliable estimates. Similarly, the Bimodal-GM distribution shows excellent performance, as the empirical mean (2.0130) and standard deviation (2.7228) are very close to the theoretical values, with an MC error of 0.0640.

The Wald(3,2) distribution shows slightly bigger differences, with the mean value (3.0956) and standard deviation (3.8450) not matching the expected values (3.0000 and 3.6742) very closely. The Monte Carlo error of 0.1112 shows less precision, which suggests difficulties of taking samples from its heavy-tailed nature. The Sinu-Exp distribution has a slightly higher empirical mean (4.1778) and standard deviation (3.9803) than the expected values (4.0871 and 3.9578). It also has a larger Monte Carlo error of 0.1240, showing how hard it is to accurately represent its wave-like pattern.

Table 3: Theoretical and empirical summary statistics for each distribution based on 5,000 samples

Sample Size	Theoretical/Empirical	Posterior Mean	SD	95% Credible set		MC Error
				2.5%	97.5%	
Beta(2,6)	Theoretical	0.250000	0.144340			
	Empirical	0.251141	0.145339	0.034600	0.586000	0.002411
Wald(2,6)	Theoretical	3.000000	3.674240			
	Empirical	3.095550	3.844950	0.329000	14.075600	0.111160
Sinu-Exp	Theoretical	4.087144	3.957808			
	Empirical	4.177800	3.980290	0.403700	15.023700	0.123960
Bimodal-GM	Theoretical	2.000000	2.738613			
	Empirical	2.010310	2.722750	-1.743800	7.252800	0.064010

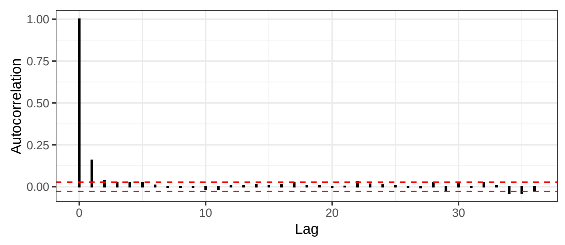
3.2 Mixing and Sampling Efficiency

The mixing and the sampling efficiency of the slice sampling algorithm were assessed by the ACF plots in Figure 6 and ESS values in Table 4. These metrics collectively measure the independence of successive samples and the performance of the algorithm on different distributions.

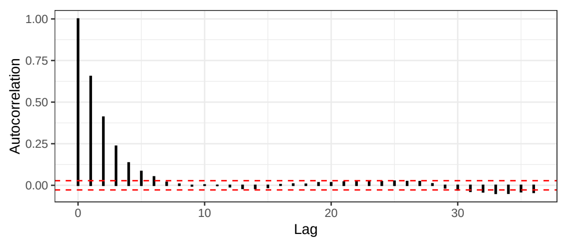
For the Beta(2,6) distribution, the autocorrelation drops off very quickly, reaching near zero by lag 2, and the ESS is the highest at 3,635 out of 5,000 samples (72.7%). These results reflect excellent mixing and efficiency, with minimal dependence between samples. Similarly, the Bimodal-GM distribution demonstrates good performance, with autocorrelation falling below the significance threshold by lag 4 and an ESS of 1,809 (36.2%). This indicates that the algorithm handles the multimodal structure effectively.

In contrast, the Wald(3,2) distribution mixes slower, with the autocorrelation decaying to nearly zero around lag 8, and a much smaller ESS of 1,196 (23.9%). This relative inefficiency is likely due to the heavy-tailed nature of the distribution. The slowest decay in the autocorrelation function, extending to lag 12 with several significant values, corresponds to the Sinu-Exp distribution, which also has the smallest ESS at 1,031 (20.6%). These results show clearly the difficulties of efficient sampling from distributions with oscillatory behavior.

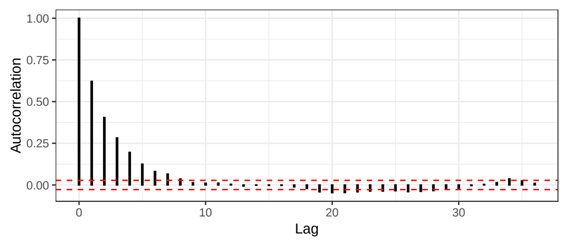
Overall, both ACF and ESS analyses show that the slice sampling algorithm has high efficiency and good mixing for a simpler distribution like Beta(2,6) but does reasonably well for more complicated distributions, with reduced efficiencies in cases involving heavy tails or oscillations.



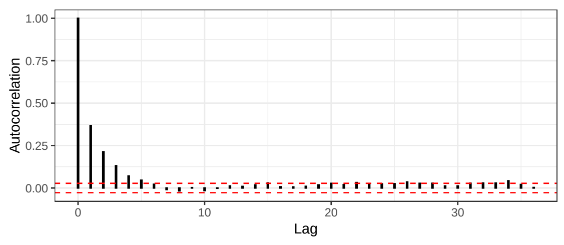
(a) Beta(2,6)



(b) Wald(3,2)



(c) Sinu-Exp



(d) Bimodal-GM

Figure 6: ACF plots for the Beta(2,6), Wald(3,2), Sinu-Exp, and Bimodal-GM distributions. The plots illustrate the mixing efficiency of the slice sampling algorithm, with rapid decay in autocorrelation for Beta(2,6) and Bimodal-GM distributions, and slower decay for the Wald(3,2) and Sinu-Exp distributions.

Table 4: Effective Sample Size (ESS) for each distribution out of 5,000 samples.

Distribution	ESS
Beta(2,6)	3634.943
Wald(3,2)	1196.477
Sinu-Exp	1031.080
Bimodal-GM	1809.441

4 Discussion

These results show the strength of the slice sampling algorithm over a wide range of different distributions. For relatively simple bounded distributions, such as the Beta(2, 6), the performance is exemplary: empirical and theoretical CDFs and PDFs align with one another nearly perfectly, with high ESS and rapidly decaying autocorrelation. Similarly, the Bimodal-GM distribution was approximated well, which again pointed out that the algorithm did not get stuck in a single mode while exploring multimodal distributions.

However, for more challenging distributions such as Wald(3,2) and Sinu-Exp, the algorithm struggled. In particular, the Wald(3,2) distribution had a heavy tail that slowed mixing, reflected in the lower ESS and persistent autocorrelation. Similarly, the Sinu-Exp distribution, due to its oscillatory behavior, showed smoothing in the sampled PDFs and a slower decay in autocorrelation. These results illustrate how the complexity of the distribution affects the efficiency of sampling and indicate that the algorithm might need tuning–for example, a slice width adjustment–for the best performance in such cases.

In all, the decreasing L2 distances with an increase in sample size confirm convergence. Empirical comparisons and efficiency metrics confirm the accuracy and reliability of the slice sampling algorithm. Future work can be directed to adaptive tuning strategies or hybrid methods to further improve the performance for distributions that are challenging, such as heavy-tailed or highly oscillatory ones. Despite these limitations, the algorithm performed robustly across all tested scenarios, showing its utility as a flexible tool for sampling from diverse distributions.

5 References

Neal, Radford M. 2003. "Slice Sampling." *The Annals of Statistics* 31 (3): 705–67.

Posit team. 2024. *RStudio: Integrated Development Environment for r.* Boston, MA: Posit Software, PBC. http://www.posit.co/.

R Core Team. 2024. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

6 Appendix

6.1 Slice Sampling Algorithm

6.1.1 Doubling Procedure

```
doubling <- function( f, x0, y, w, p ){
  U <- runif( 1 )
  L <- x0 - w * U
  R <- L + w
  K <- p
  while ( K > 0 && ( y < f(L) || y< f(R) ) ) {
    V <- runif( 1, 0, 1 )
    if( V < 1/2 ){
      L <- L - ( R - L )
    }else{
      R <- R + ( R - L )
    }
    K <- K-1
  }
  return( c(L, R) )
}
```

6.1.2 Test Acceptance Procedure

```
test.acceptance <- function( f, x0, x1, y, w, L, R ){
  L.hat <- L
  R.hat <- R
  D = FALSE
  while ( R.hat  - L.hat > 1.1 * w ) {
    M <- ( L.hat + R.hat ) / 2
    if ( ( x0 < M && x1 >= M )  || ( x0 >= M && x1 < M ) ){
      D <- TRUE
    }
    if ( x1 < M ){
      R.hat <- M
    }else{
      L.hat <- M
    }
    if ( D &&  y >= f( L.hat ) && y >= f( R.hat ) ){
      return( FALSE )
    }
  }
  return( TRUE )
}
```

6.1.3 Shrinkage Procedure

```
shrinkage <- function( f, x0, L, R, y, w ){
  L.bar <- L
  R.bar <- R
  while(TRUE) {
    U <- runif(1)
    x1 <- L.bar + U * ( R.bar - L.bar )
    if ( y < f( x1 ) && test.acceptance( f, x0, x1, y, w, L.bar, R.bar )) {
        return( x1 )
      }
    if ( x1 < x0 ){
      L.bar <- x1
    }else{
      R.bar <- x1
    }
  }
}
```

6.1.4 Slice Sampler

```
slice.sampling <- function( f, x0, w, p, n.samples ){
    samples <- numeric(n.samples)
    samples[1] <- x0
    for ( i in 2:n.samples ){
        y <- runif( 1, 0, f( x0 ) )
        L_R <- doubling( f, x0, y, w, p )
        L <- L_R[1]
        R <- L_R[2]
        x1 <- shrinkage( f, x0, L, R, y, w )
        samples[i] <- x1
        x0 <- x1
    }
    return ( samples )
}
```

6.2 Metric Functions

6.2.1 L2 Distance

```
l2.distance<- function(samples, theoretical.cdf, l.limit, u.limit){
  empirical.cdf <- ecdf(samples)
  x.vals <- seq(l.limit, u.limit, length.out = 1000)
  empirical.values <- empirical.cdf(x.vals)
  theoretical.values <- theoretical.cdf(x.vals)
  squared.diff <- (empirical.values - theoretical.values)^2
  L2.distance <- sum(squared.diff) * (x.vals[2] - x.vals[1])
  return(L2.distance)
}
```

6.2.2 CDF Plot Function

```
cdf.plot <- function(samples, theoretical.cdf, l.limit, u.limit){
  empirical.cdf <- ecdf(samples)
  x.vals <- seq(l.limit, u.limit, length.out = 1000)
  empirical.values <- empirical.cdf(x.vals)
  theoretical.values <- theoretical.cdf(x.vals)
  plot.data <- data.frame(
  x = rep(x.vals, 2),
  CDF = c(theoretical.values, empirical.values),
  Type = rep(c("Theoretical CDF", "Empirical CDF"),each = length(x.vals))
  )
  p<-ggplot(plot.data, aes(x = x, y = CDF, color = Type)) +
  geom_line(linewidth = 1.2) +
  labs(
    x = "x",
    y = "CDF",
    color = "CDF Type"
  ) +
  scale_color_manual(values = c("black", "red")) +
  theme_bw() +
  theme(
    legend.position = "bottom",
    legend.title = element_blank()
  )
  return(plot = p)
}
```

6.2.3 PDF Plot Function

```
pdf.plot <- function(sample.density, true.density, x.vals){
  true.density.data <- data.frame(
    x = x.vals,
    density = true.density,
    Type = "True Density"
  )
  sample.density.data <- data.frame(
    x = sample.density$x,
    density = sample.density$y,
    Type = "Slice Sample Density"
  )
  hist.data <- data.frame(x = samples)
  p <- ggplot() +
      geom_histogram(
        data = hist.data, aes(x = x, y = ..density..),
        bins = 30, fill = "gray", color = "black", alpha = 0.6
      ) +
      geom_line(
        data = true.density.data, aes(x = x, y = density, color = Type),
        size = 1
      ) +
      geom_line(
        data = sample.density.data, aes(x = x, y = density, color = Type),
        linetype = "dashed", size = 1
      ) +
      labs(
        x = "x",
        y = "Density",
        color = "Legend"
      ) +
      scale_color_manual(values = c("red", "black")) +
      theme_bw()+
      theme(
        legend.position = "bottom",
        legend.title = element_blank()
        )+
    coord_cartesian(xlim = c(min(x.vals), max(x.vals)))
  return(plot = p)
}
```

6.2.4 ACF Plot

```
acf.plot <- function(samples){
  acf.result <- acf(samples, plot = FALSE)
  acf.data <- data.frame(
    Lag = acf.result$lag[,1,1],
    Autocorrelation = acf.result$acf[,1,1]
  )
  p<-ggplot(acf.data, aes(x = Lag, y = Autocorrelation)) +
    geom_bar(stat = "identity", fill = "gray", color = "black", width = 0.1) +
    geom_hline(yintercept = c(-1.96 / sqrt(length(samples)),
                              1.96 / sqrt(length(samples))),
               linetype = "dashed", color = "red") +
    theme_bw() +
    labs(x = "Lag", y = "Autocorrelation")
  return(plot=p)
}
```

6.2.5 Posterior Summary and ESS

```
post.ess<-function(samples){
  mcmc.samples <- as.mcmc(samples)
  s.summary<-summary(mcmc.samples)
  s.size<-effectiveSize(mcmc.samples)
  return(list(post = s.summary, ess = s.size))
}
```

6.3 Sampling

6.3.1 Sampling From Beta(2,6)

```
alpha <- 2
beta <- 6
target.density <- function(x) {
  if (x < 0 || x > 1) return(0)
  dbeta(x, shape1 = alpha, shape2 = beta)
}
theoretical.cdf <- function(x) pbeta(x, shape1 = alpha, shape2 = beta)
x0 <- 0.5
w <- 0.1
p <- 10
n.samples <- 5000
```

```
samples = slice.sampling(target.density, x0, w, p, n.samples)
mean.samples <- mean(samples)
sd.samples <- sd(samples)
#statistics and l2 distance
l.limit <- 0
u.limit <- 1
l2.samples <- l2.distance(samples, theoretical.cdf, 0, 1)
cat("l2: ", l2.samples, "\n")
#cdf plot
cdf.samples <- cdf.plot(samples, theoretical.cdf, 0, 1)
print(cdf.samples)
#pdf plot
x.vals <- seq(l.limit, u.limit, length.out = 1000)
true.density <- dbeta(x.vals, shape1 = 2, shape2 = 6)
sample.density <- density(samples, from = l.limit, to = u.limit)
pdf.samples<-pdf.plot(sample.density, true.density, x.vals)
print(pdf.samples)
#acf plot
acf.samples<-acf.plot(samples)
print(acf.samples)
#posterior summary ess
post.ess.samples<-post.ess(samples)
print(post.ess.samples)
#theoretical statistics
cat("t mean", (alpha/(alpha+beta)),"\n")
cat("t sd", sqrt( ( alpha*beta ) / ( (alpha+beta)^2 * (alpha + beta +1) ) ),"\n")
```

6.3.2 Sampling From Wald(3,2)

```
mu<-3
lambda <- 2
target.density <- function(x) {
  if (x <= 0) return(0)
  sqrt(lambda / (2 * pi * x^3)) * exp(-lambda * (x - mu)^2 / (2 * mu^2 * x))
}
theoretical.cdf <- function(x) pwald(x, mu, lambda)
x0 <- 3
w <- 0.1
p <- 10
n.samples <- 5000
samples = slice.sampling(target.density, x0, w, p, n.samples)
mean.samples <- mean(samples)
sd.samples <- sd(samples)
#statistics and l2 distance
l.limit <- 0
u.limit <- 15
l2.samples <- l2.distance(samples, theoretical.cdf, 0, 15)
cat("l2: ", l2.samples, "\n")
#cdf plot
cdf.samples <- cdf.plot(samples, theoretical.cdf, 0, 15)
print(cdf.samples)
#pdf plot
x.vals <- seq(l.limit, u.limit, length.out = 1000)
true.density <- dwald(x.vals, mu = 3, lambda = 2)
samples <- samples[samples >= l.limit & samples <= u.limit]
sample.density <- density(samples, from = l.limit, to = u.limit)
pdf.samples<-pdf.plot(sample.density, true.density, x.vals)
print(pdf.samples)
#acf plot
acf.samples<-acf.plot(samples)
print(acf.samples)
#posterior summary ess
post.ess.samples<-post.ess(samples)
print(post.ess.samples)
#theoretical statistics
cat("t mean: ", mu, "\n")
cat("t sd: ", sqrt(mu^3/lambda), "\n")
```

6.3.3 Sampling From Sinu-Exp

```
target.density <- function(x) {
  ifelse(x <= 0, 0, abs(sin(x)) * exp(-x / 4))
}
```

```
normalizing.constant <- integrate(
  function(x) abs(sin(x)) * exp(-x / 4),
  lower = 0, upper = Inf
)$value
theoretical.pdf <- function(x) {
  ifelse(x <= 0, 0, (abs(sin(x)) * exp(-x / 4)) / normalizing.constant)
}
theoretical.cdf <- function(x) {
  sapply(x, function(xi) {
    if (xi <= 0) {
      0
    } else {
      integrate(theoretical.pdf, lower = 0, upper = xi)$value
    }
  })
}
x0 <- 0.5
w <- 0.1
p <- 10
n.samples <- 5000
samples = slice.sampling(target.density, x0, w, p, n.samples)
mean.samples <- mean(samples)
sd.samples <- sd(samples)
#statistics and l2 distance
l.limit <- 0
u.limit <- 15
l2.samples <- l2.distance(samples, theoretical.cdf, 0, 15)
cat("l2: ", l2.samples, "\n")
#cdf plot
cdf.samples <- cdf.plot(samples, theoretical.cdf, 0, 15)
print(cdf.samples)
#pdf plot
x.vals <- seq(l.limit, u.limit, length.out = 1000)
true.density <- theoretical.pdf(x.vals)
samples <- samples[samples >= l.limit & samples <= u.limit]
sample.density <- density(samples, from = l.limit, to = u.limit)
pdf.samples<-pdf.plot(sample.density, true.density, x.vals)
print(pdf.samples)
#acf plot
acf.samples<-acf.plot(samples)
print(acf.samples)
#posterior summary ess
post.ess.samples<-post.ess(samples)
print(post.ess.samples)
#theoretical statistics
t.mean <- integrate(
  function(x) x * theoretical.pdf(x),
  lower = 0,
  upper = Inf
)$value
second.moment <- integrate(
  function(x) x^2 * theoretical.pdf(x),
  lower = 0,
  upper = Inf,
  subdivisions = 1000
)$value
t.variance <- second.moment - (theoretical.mean)^2
t.sd <- sqrt(theoretical.variance)
cat("t mean: ", t.mean, "\n")
cat("t sd: ", t.sd, "\n")
```

6.3.4 Sampling From Bimodal-GM

```
w1 <- 0.6
mu1 <- 0
sigma1 <- 1
w2 <- 0.4
mu2 <- 5
sigma2 <- 1.5
gaussian.pdf <- function(x, mu, sigma) {
  (1 / (sqrt(2 * pi) * sigma)) * exp(-0.5 * ((x - mu) / sigma)^2)
}
```

```
target.density<- function(x) {
  w1 * gaussian.pdf(x, mu1, sigma1) + w2 * gaussian.pdf(x, mu2, sigma2)
}
gaussian.cdf <- function(x, mu, sigma) {
  pnorm(x, mean = mu, sd = sigma)
}
theoretical.cdf <- function(x) {
  w1 * gaussian.cdf(x, mu1, sigma1) + w2 * gaussian.cdf(x, mu2, sigma2)
}
x0 <- 0.5
w <- 0.1
p <- 10
n.samples <- 5000
samples = slice.sampling(target.density, x0, w, p, n.samples)
mean.samples <- mean(samples)
sd.samples <- sd(samples)
#statistics and l2 distance
l.limit <- -10
u.limit <- 15
l2.samples <- l2.distance(samples, theoretical.cdf, -10, 15)
cat("l2: ", l2.samples, "\n")
#cdf plot
cdf.samples <- cdf.plot(samples, theoretical.cdf, -10, 15)
print(cdf.samples)
#pdf plot
x.vals <- seq(l.limit, u.limit, length.out = 1000)
true.density <- target.density(x.vals)
samples <- samples[samples >= l.limit & samples <= u.limit]
sample.density <- density(samples, from = l.limit, to = u.limit)
pdf.samples<-pdf.plot(sample.density, true.density, x.vals)
print(pdf.samples)
#acf plot
acf.samples<-acf.plot(samples)
print(acf.samples)
#posterior summary ess
post.ess.samples<-post.ess(samples)
print(post.ess.samples)
#theoretical statistics
t.mean <- w1 * mu1 + w2 * mu2
t.variance <- w1 * (sigma1^2 + mu1^2) + w2 * (sigma2^2 + mu2^2) - gmm.mean^2
t.sd <- sqrt(gmm.variance)
cat("t mean: ", t.mean, "\n")
cat("t sd: ", t.sd, "\n")
```