Sampling from the Banana Distribution

Will Gertsch

12/18/2020

```
# setup
source('~/BIOS213/final_functions.R')
set.seed(2020)
N = 1e5
```

Introduction

The banana distribution was introduced in Haario (1999) as an example distribution for testing sampling algorithms. It has the advantage of being easy to sample from directly, so it is easy to check if the sampling algorithm is working properly.

In this report, I will sample from a specific banana distribution directly, using the Metropolis algorithm, and with Gibbs sampling. The kernel of the distribution is

$$\pi(x, y) \propto e^{-x^2/2} e^{-\frac{1}{2}\left(y-2(x^2-5)\right)^2}$$

Direct transformation method

Per Haario (1999), this banana distribution can be constructed by "twisting" a bivariate normal distribution as follows. Let

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \sim N(0, I_2)$$

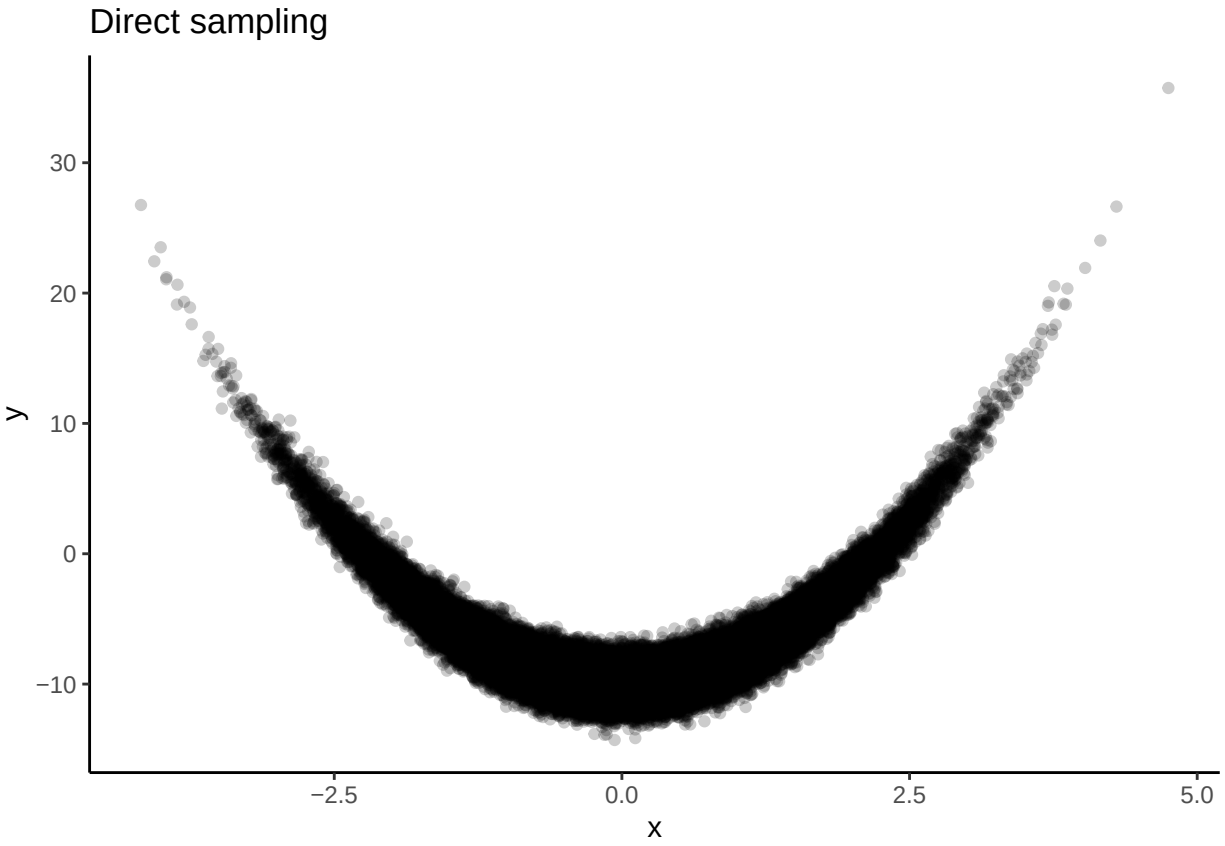
and apply the change of variables transformation $x = x_0$ and $y = y_0 + 2x_0^2 - 10$ using the density function of the bivariate normal. After some algebra, the density function is

$$\pi(x, y) = \frac{1}{2\pi} e^{-\frac{1}{2}\left[x^2 + (y-2(x^2-5)^2\right]}$$

where with $x, y \in \mathbb{R}$, which is the same as the desired banana density.

Practically, this means that a sample from the banana distribution can be obtained by drawing a sample from $N(0, I_2)$ and applying the above transformation. The following R code samples using this method.

```
# sample by transformation
samples_direct <- sample_banana(N) # 0.386s
ggplot() +
  geom_point(aes(x = samples_direct[,1], y = samples_direct[,2]), alpha = 0.2) +
  theme_classic() +
  xlab("x") + ylab("y") + ggtitle("Direct sampling")
```

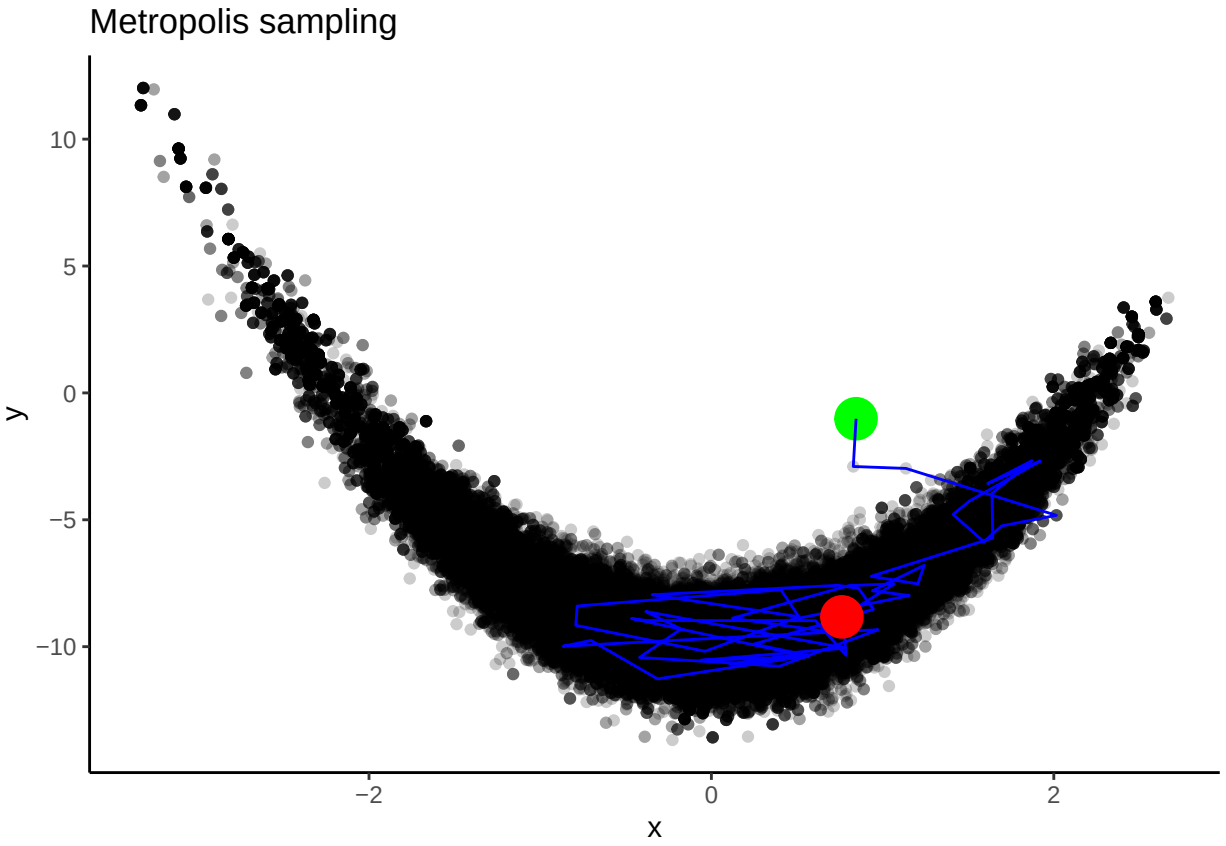


Metropolis algorithm

The Metropolis algorithm is a general purpose algorithm for drawing samples from a distribution using markov chains and can be used even if the distribution's normalizing constant is unknown. The algorithm works by randomly walking through the sample space and approximates the distribution by being more likely to "step" into samples that have a higher probability.

Since the Metropolis algorithm tries to move to points where the probability is high, it must frequently reject possible steps where the probability would decrease. This accept/reject process means the Metropolis algorithm is substantially more computationally intensive than sampling directly. The following R code takes at least 14 seconds to run on my machine compared to less than a second for direct sampling.

```
start = c(rnorm(1), rnorm(1))
samples_met <- metropolis_banana(N, start) # 14s
path_1st_200(samples_met, "Metropolis sampling")
```



The path of the first 200 samples are overlaid to give an idea of the random walk of the algorithm. The initial guess sample is represented by the green dot while the red dot is the 200th sample. Note that the Metropolis algorithm seems to have trouble sampling from the right tail of the distribution.

Gibbs Sampler

Another way to sample from the banana distribution is to sample from the conditional distributions. This approach results in a Markov chain that randomly walks through the sample space of the distribution similar to the Metropolis algorithm. This is known as Gibbs sampling and is a popular approach for sampling from a distribution where the conditionals are easy to compute.

We can derive $\pi(y|x)$ analytically.

$$\pi(y|x) \propto \frac{\pi(x,y)}{\pi(x)}$$

$$\pi(x) \propto \int_{-\infty}^{\infty} \pi(x,y)dy = e^{-x^2/2} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(y-2(x^2-5))^2} dy = e^{-x^2/2} \cdot \sqrt{2\pi}$$

Therefore, the conditional distribution is

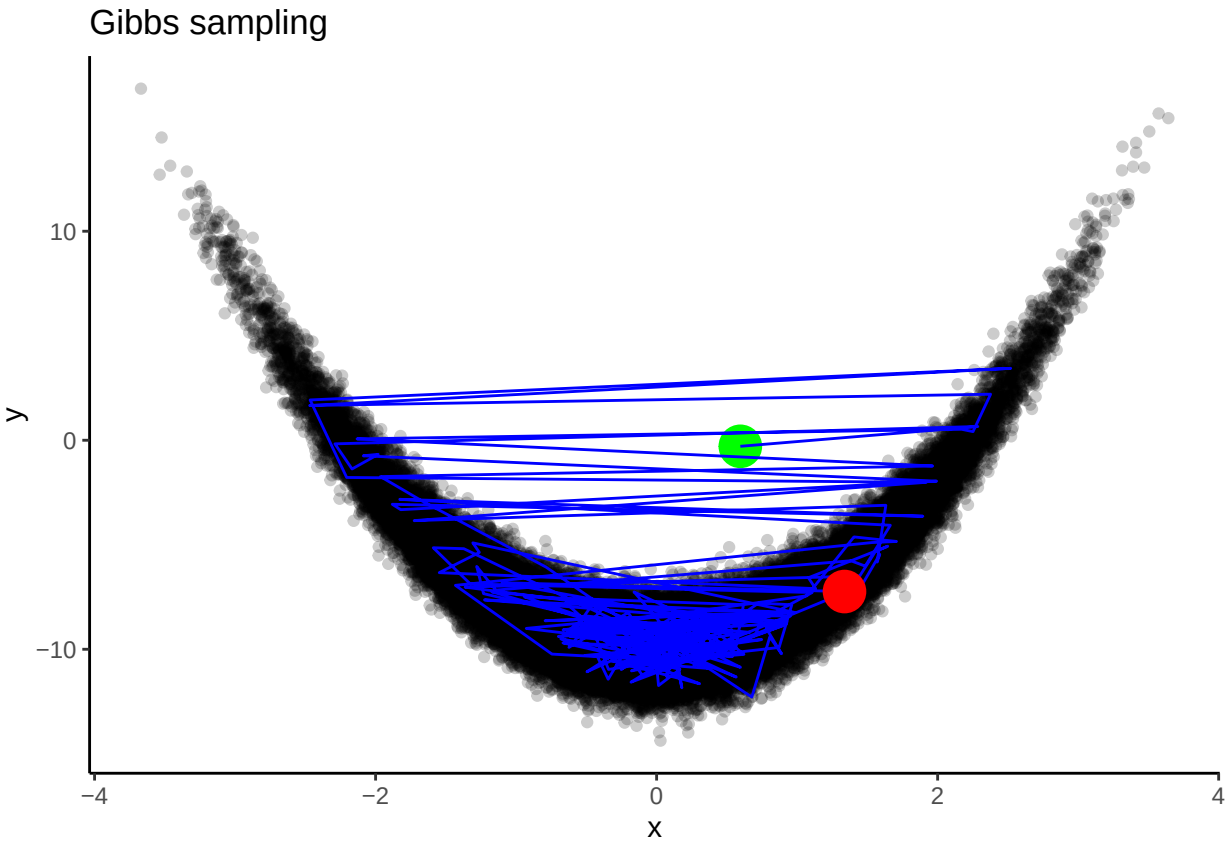
$$\pi(y|x) \propto \frac{\pi(x,y)}{\pi(x)} \propto e^{-\frac{1}{2}(y-2(x^2-5))^2}$$

which is a normal distribution with mean $2(x^2-5)$ and variance 1.

Finding $\pi(x|y)$ analytically is more difficult, but can also be estimated numerically. The approach I used was to approximate the cumulative density function $F(X|Y)$ and then use fact that $x = F^{-1}(u)$ where $u \sim U(0,1)$ to draw a sample x from $X|Y$.

Gibbs sampling is much faster than the Metropolis algorithm for sampling from the banana distribution and gives a much different random walk.

```
# sample by Gibbs
start <- c(rnorm(1), rnorm(1))
samples_gibbs <- gibbs_banana(N, start) # 2.26s
path_1st_200(samples_gibbs, "Gibbs sampling")
```



Note that the Gibbs random walk is more back and forth as compared to the Metropolis algorithm's smaller steps. Gibbs sampling also does a better job of capturing the right tail of the distribution compared to Metropolis.

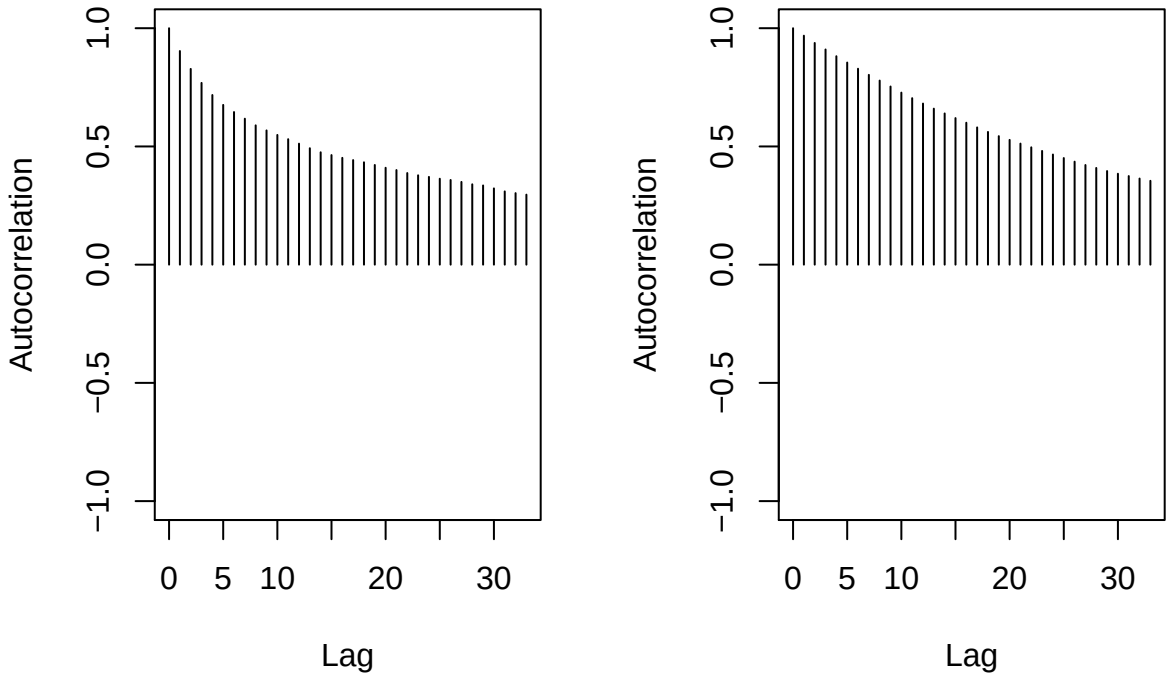
Comparison of different jump sizes

One important parameter used in the Metropolis algorithm is the variance of the proposal distribution, w, which determines how big the steps will be in the random walk. To see which value of w is best, I compared mixing and convergence between $w = 0.5$, $w = 1.0$, and $w = 2.0$. I used the `coda` library for the following plots and diagnositics. For all chains, I drew 10,000 samples and discarded the first 5,000 for burn-in.

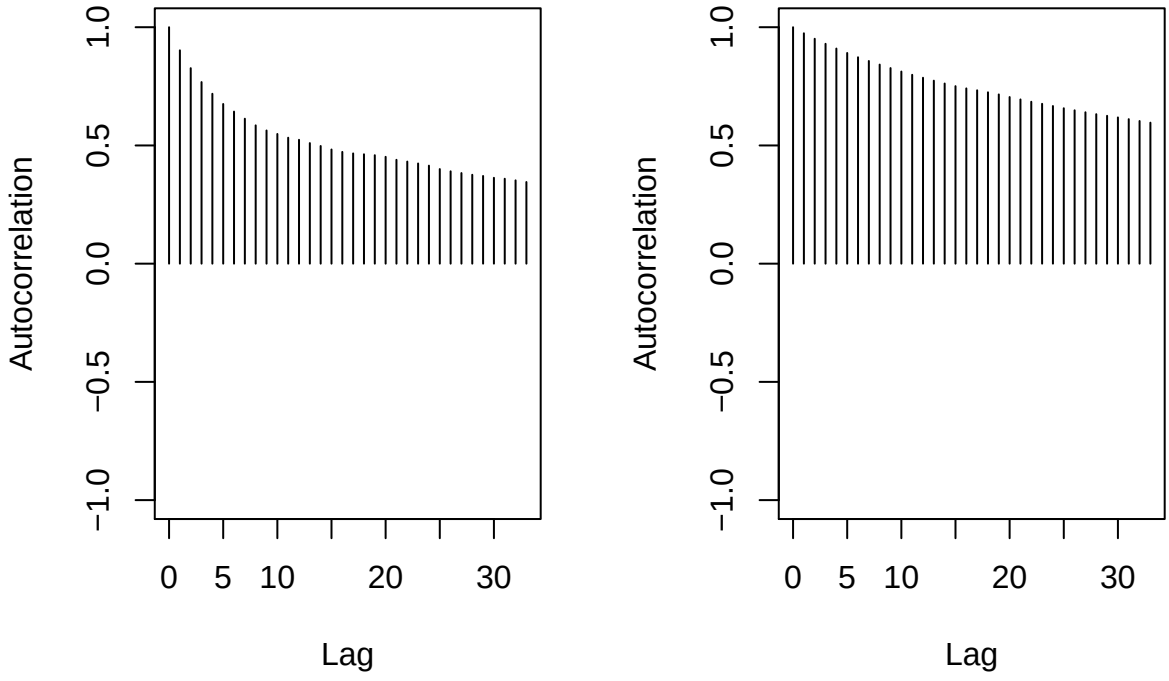
Mixing is roughly the speed at which the markov chain converges to the target distribution. It is possible to measure this speed by looking at the correlation between individual steps. If there is high autocorrelation, then the chain will mix slowly. Below are the autocorrelation plots for each w with the chain starting in the same place.

```
start = c(rnorm(1), rnorm(1))
samples_met1 <- as.mcmc(metropolis_banana(1e4, start, w = 0.5)[5000:1e4,])
samples_met2 <- as.mcmc(metropolis_banana(1e4, start, w = 1)[5000:1e4,])
```

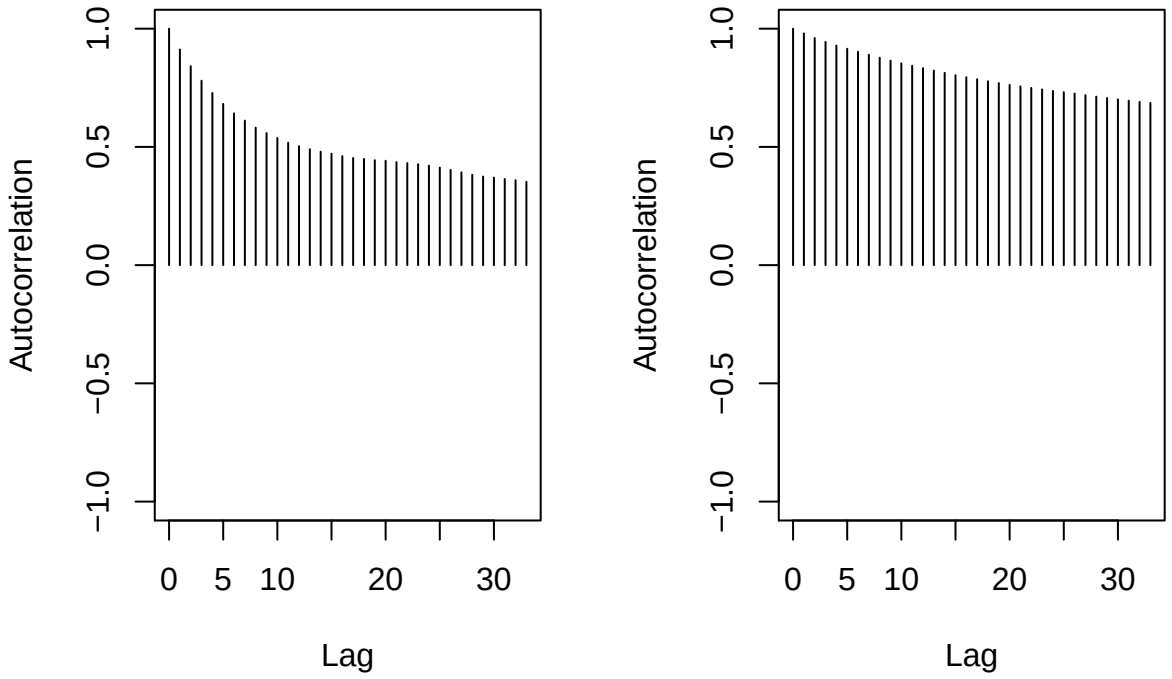
```r
samples_met3 <- as.mcmc(metropolis_banana(1e4, start, w = 2)[5000:1e4,])

autocorr.plot(samples_met1) # w = 0.5
```




```r
autocorr.plot(samples_met2) # w = 1
```




```r
autocorr.plot(samples_met3) # w = 2
```

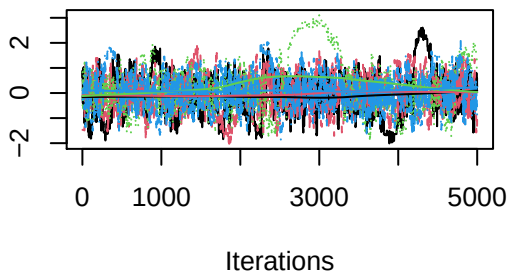


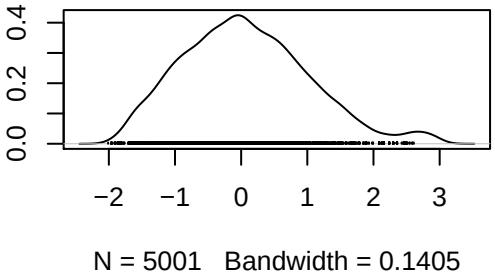
$w = 2$ seems to have the lowest autocorrelation by a tiny bit, but there seems to be a lot in all cases. For future analysis, it might be a good idea to implement methods to improve mixing.

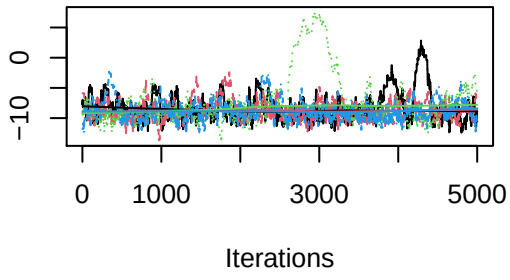
To assess convergence, I ran 4 chains with distinct starting locations to see if all the chains would end up in the same place and to see what the effect of w would be.

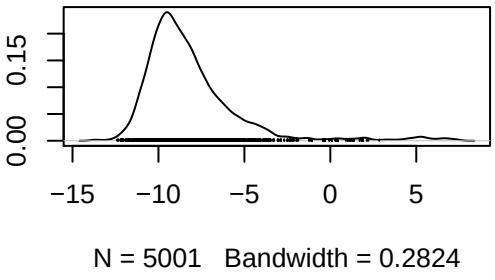
```
# starting from 4 different points
start1 <- c(-3, 20) # left tail
start2 <- c(3, 20) # right tail
start3 <- c(0, 10) # focus point
start4 <- c(0, -5) # high density

# start position 1 with w = 0.5
chain_w5_s1 <- as.mcmc(metropolis_banana(1e4, start1, w = 0.5)[5000:1e4,])
chain_w5_s2 <- as.mcmc(metropolis_banana(1e4, start2, w = 0.5)[5000:1e4,])
chain_w5_s3 <- as.mcmc(metropolis_banana(1e4, start3, w = 0.5)[5000:1e4,])
chain_w5_s4 <- as.mcmc(metropolis_banana(1e4, start4, w = 0.5)[5000:1e4,])
# etc
chain_w1_s1 <- as.mcmc(metropolis_banana(1e4, start1, w = 1)[5000:1e4,])
chain_w1_s2 <- as.mcmc(metropolis_banana(1e4, start2, w = 1)[5000:1e4,])
chain_w1_s3 <- as.mcmc(metropolis_banana(1e4, start3, w = 1)[5000:1e4,])
chain_w1_s4 <- as.mcmc(metropolis_banana(1e4, start4, w = 1)[5000:1e4,])
chain_w2_s1 <- as.mcmc(metropolis_banana(1e4, start1, w = 2)[5000:1e4,])
chain_w2_s2 <- as.mcmc(metropolis_banana(1e4, start2, w = 2)[5000:1e4,])
chain_w2_s3 <- as.mcmc(metropolis_banana(1e4, start3, w = 2)[5000:1e4,])
chain_w2_s4 <- as.mcmc(metropolis_banana(1e4, start4, w = 2)[5000:1e4,])

# combine chains into a single object for plotting
combined_w5 <- mcmc.list(chain_w5_s1, chain_w5_s2, chain_w5_s3, chain_w5_s4)
combined_w1 <- mcmc.list(chain_w1_s1, chain_w1_s2, chain_w1_s3, chain_w1_s4)
combined_w2 <- mcmc.list(chain_w2_s1, chain_w2_s2, chain_w2_s3, chain_w2_s4)
```

```
plot(combined_w5)
```

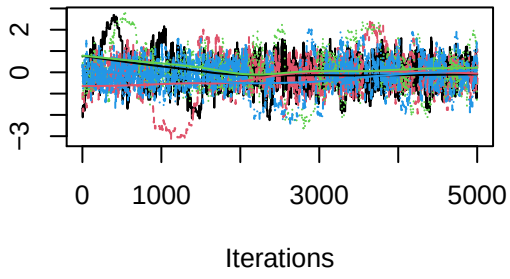


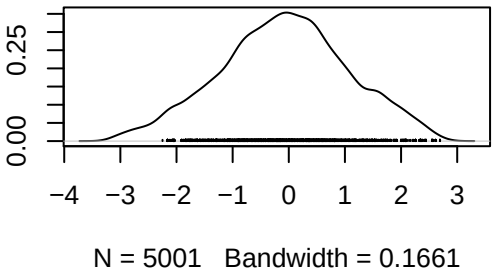


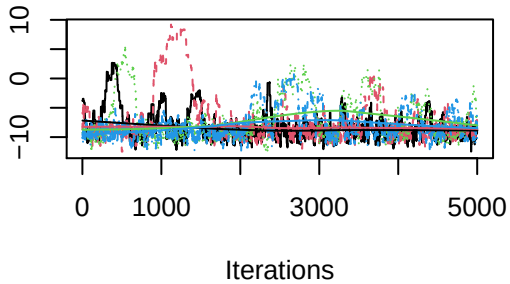


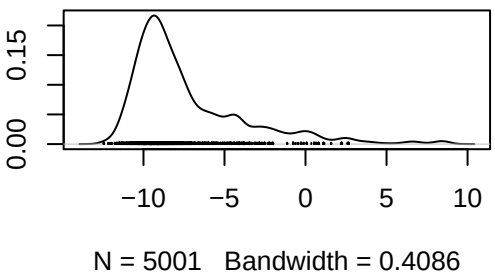



```
plot(combined_w1)
```

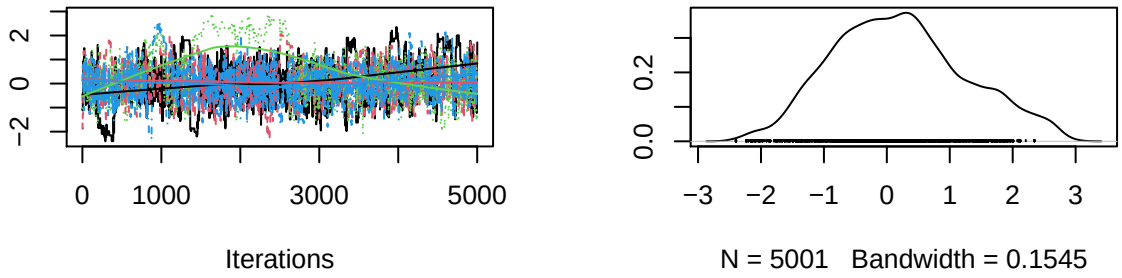


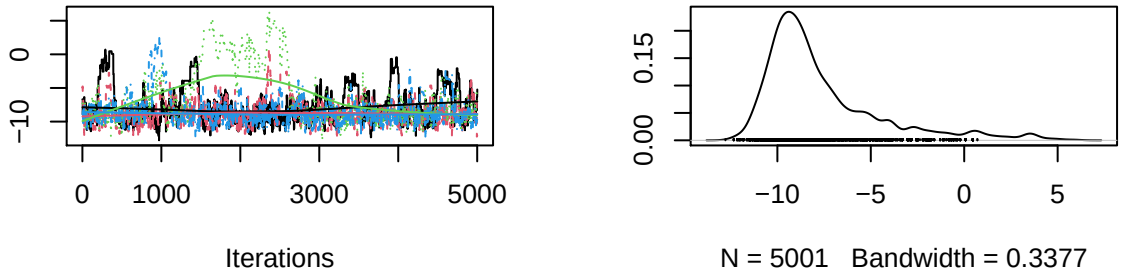







```
plot(combined_w2)
```

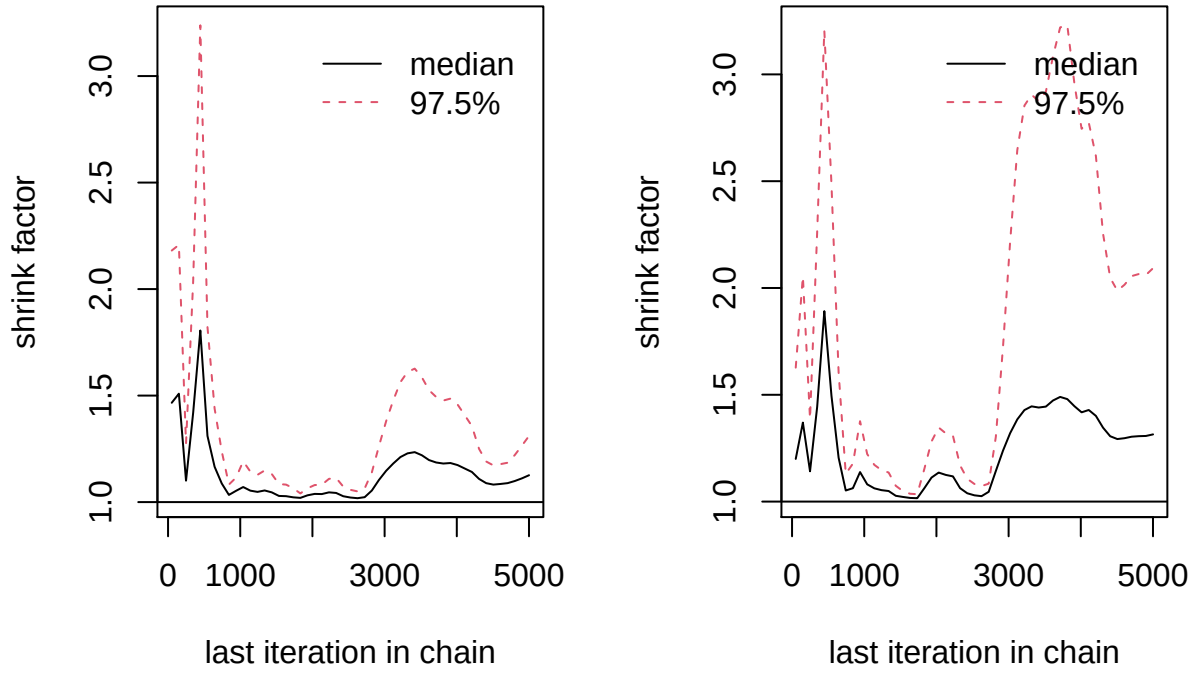




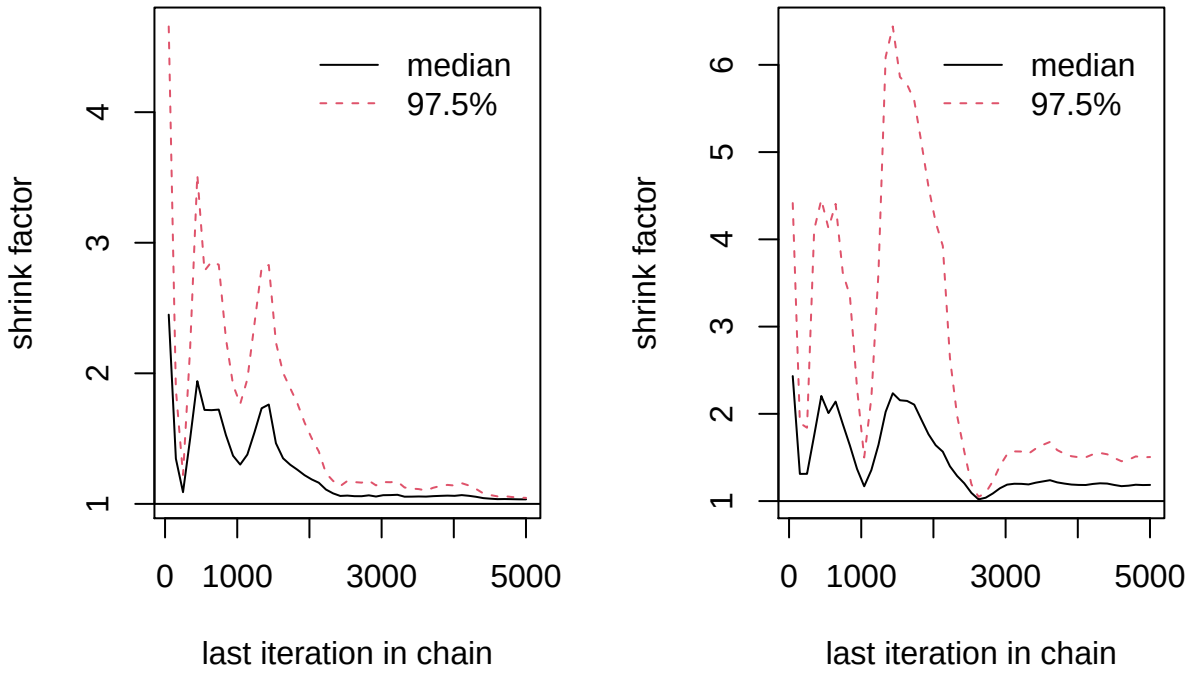
From the trace plots, it seems that $w = 2$ has the worst convergence properties, while $w = 0.5$ and $w = 1$ were about the same.

For more information, I also used Gelman and Rubin's convergence diagnostic, which works by comparing the variance between chains and within chains. The result is a "shrink factor" that approaches 1 when the chain converges. Plotting the shrink factor over time will give an idea of how quickly the chain converges.

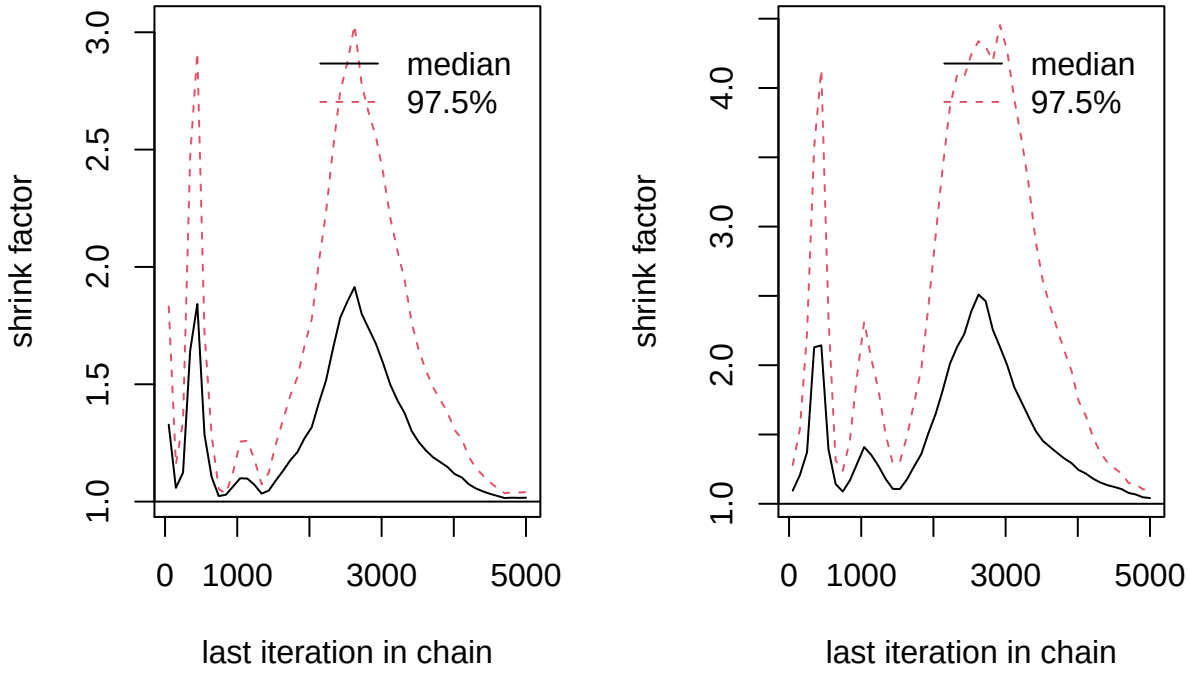
```
gelman.plot(combined_w5)
```



```
gelman.plot(combined_w1)
```




```
gelman.plot(combined_w2)
```



Here it seems that $w = 2$ is the best choice because the shrinkage factor seems to approach 1 after 5000 samples after burn-in. However, it seems erratic so $w = 1$ might be a better choice because it stabilizes after about 3000 samples after burn-in.

There is also a numerical version, which is nice because it gives a multivariate version of the scale factor.

```r
gelman.diag(combined_w5)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.13       1.31
## [2,]       1.31       2.09
##
## Multivariate psrf
##
## 1.19
```

```r
gelman.diag(combined_w1)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.03       1.05
## [2,]       1.19       1.51
##
## Multivariate psrf
##
## 1.16
```

```r
gelman.diag(combined_w2) # convergence from last 2 is much better
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.02       1.04
## [2,]       1.04       1.10
##
## Multivariate psrf
##
## 1.04
```

$w = 2$ is the best option since all the estimated scale factors are closest to 1. It is also worth noting that this diagnostic takes into account the fact that the chain is sampling from a bivariate distribution and therefore $w = 2$ may be the best option overall even though it doesn't do as well from inspection of univariate plots.

Conclusion

In this report, I have demonstrated 3 ways of sampling from the banana distribution. The direct sampling method is probably the best because it is the fastest and most accurate. Gibbs sampling seems to be the next best option, followed by the Metropolis algorithm. Care should be taken to ensure that a good jumping size is chosen for the Metropolis algorithm to ensure good mixing and convergence.