STA 360 Lab 7: Monte Carlo Simulation

Isaac Fan

18 March, 2021

Preliminaries

Please submit a pdf copy of this lab on Sakai by Friday, March 19th at 11:59 PM. Exercises 3 and 7 will be graded for completion.

Introduction to Monte Carlo Simulation

As we've seen in previous labs, Monte Carlo simulation is a very useful technique for obtaining samples from distributions to obtain means, credible intervals, and more. It turns out, however, Monte Carlo simulation is significantly more powerful in it's ability to sample from distributions that we can't even write down! In the coming weeks, we will see some examples of these approaches that fall under the umbrella of Markov Chain Monte Carlo (MCMC) methods, notably Gibbs sampling and and Metropolis Hasting. The purpose of this lab is to take stock of what we know about Monte Carlo sampling, apply it to some interesting examples, and start taking a somewhat more formal look at simulation methods.

Applications and Examples

Simulating Directly from Posterior Distributions

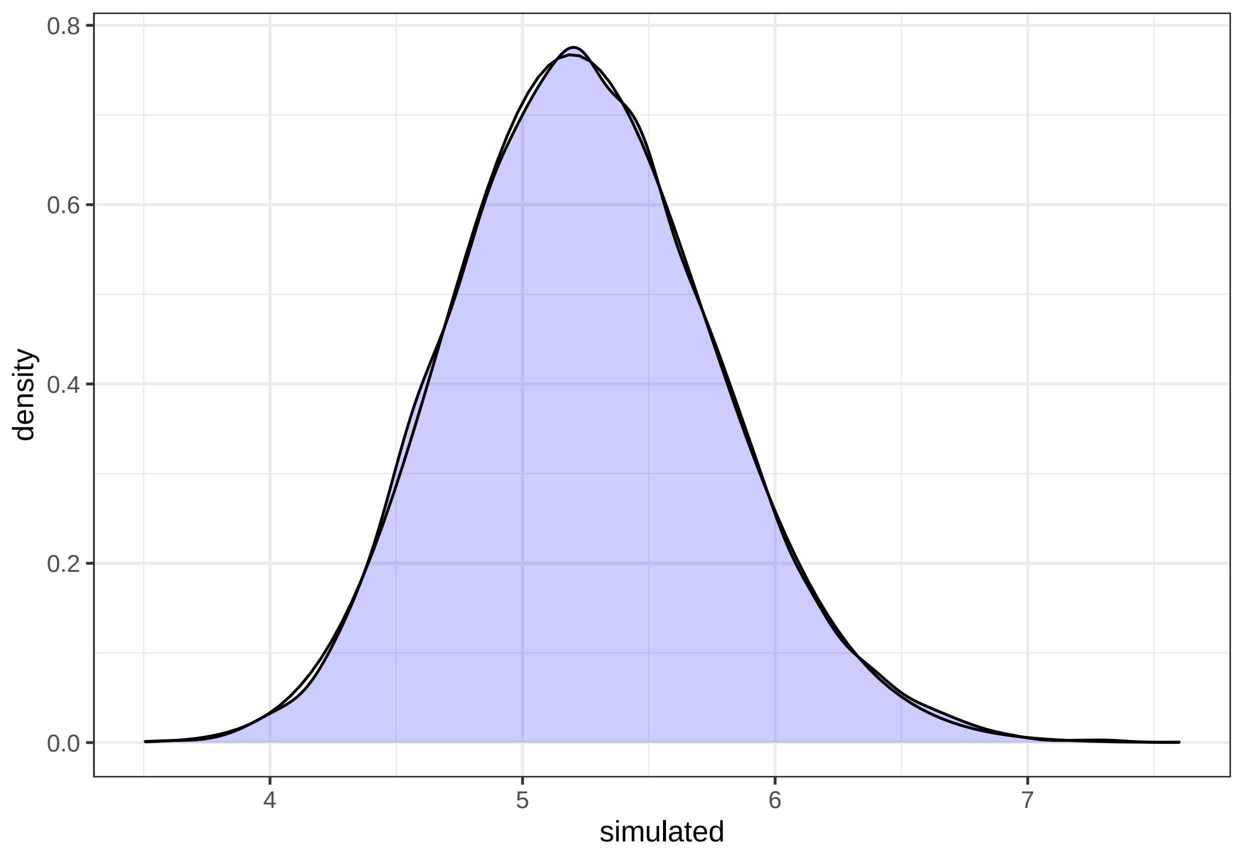
Let's start by reviewing what at this point should be a straightforward example: sampling from a posterior distribution that comes from a conjugate family. Suppose $Y_1, \dots, Y_n \sim Exp(\lambda)$, and we place a prior $\lambda \sim Ga(a, b)$. We know from a few weeks ago that

$$\lambda \mid Y_1, \dots, Y_n \sim Ga\left(a + n, b + \sum_{j=1}^{n} y_j\right)$$

```r
y <- rexp(100, 5)
```

Exercise 1 (Not for Completion) Set $\lambda = 5$, and simulate 100 values from the sampling model $Y_1, \dots, Y_n$. Then set $a = b = 1$, and sample 10,000 points from the posterior distribution $\lambda \mid Y_1, \dots, Y_n$, and make a density plot. Then plot the theoretical density for this distribution. How do these compare?

```r
lambda_samp <- data.frame(simulated = rgamma(10000, 1 + 100, 1 + sum(y)))
ggplot(data = lambda_samp, aes(x = simulated)) +
  geom_density(alpha = .2, fill = 'blue')+
  stat_function(fun = function(x) {dgamma(x, 1 + 100, 1 + sum(y))})
```



Hopefully, the above exercise involves a review of what we've done before. There are a couple of observations worth making:

1. We had a closed-form, analytical expression for the posterior (thanks to conjugacy), so we can actually write down the density. Since it's a familiar one, we can obtain summary statistics often using analytical formulas.
2. If we can't obtain an expression directly from the density, simulating is super easy since we know exactly what function to call in `R` (e.g., `rgamma`) with precisely the parameters of the posterior distribution.

Exercise 2 (Not for Completion) Compute the *exact* expectation, variance, and $\mathsf{P}(\theta \leq 3.5 \mid y_1, \ldots, y_n)$ of the posterior distribution. Compute the Monte Carlo approximations to these quantities. Compute How do they compare?

```
# True values
# Posterior mean
(1 + length(y))/(1 + sum(y))
```

```
## [1] 5.247057
```

```r
# Posterior var
(1 + length(y))/(1 + sum(y))^2
```

```
## [1] 0.2725901
```

```r
# Posterior P <= 3.5 | data
pgamma(3.5, 1 + length(y), 1 + sum(y))
```

```
## [1] 7.904835e-05
```

```r
#Estimated values
#posterior mean
mean(lambda_samp$simulated)
```

```
## [1] 5.250894
```

```r
#posterior var
var(lambda_samp$simulated)
```

```
## [1] 0.270664
```

```r
# p <= 3.5
mean(lambda_samp$simulated <= 3.5)
```

```
## [1] 0
```

Simulating Jointly with Dependencies

In the above section, we saw that all is well and good when we have access to the exact specification of our distribution. Starting with this section, we will start dealing with more challenging situations in which we may not be able to precisely describe the distribution from which we're sampling.

Suppose we have a 2 random variables X and Y with joint distribution given by $f_{XY}(x, y)$. Suppose also that we know that $Y \sim Bernoulli(0.8)$, $(X|Y = 0) \sim Beta(1, 5)$, and $(X|Y = 1) \sim Beta(5, 1)$. Our goal is to draws samples from the joint distribution f_{XY}. A sample consists of a pair (X, Y), so we need to draw two values, one for X and one for Y, to get a single observation from the joint distribution. How can we do this? Well, recall from your previous probability class that we can write:

$$f_{XY}(x, y) = f_{X|Y}(x \mid y) f_Y(y)$$

What this means is that we can draw samples from the marginal distribution of Y, which is Bernoulli here, and then using that value of Y, we can simulate from $X|Y$ to get a value for X. The resulting pair (X, Y) is a draw from the joint distribution f_{XY}.

Exercise 3

(a) Sample 10,000 observations from the marginal distribution $f_Y(y)$ of Y.

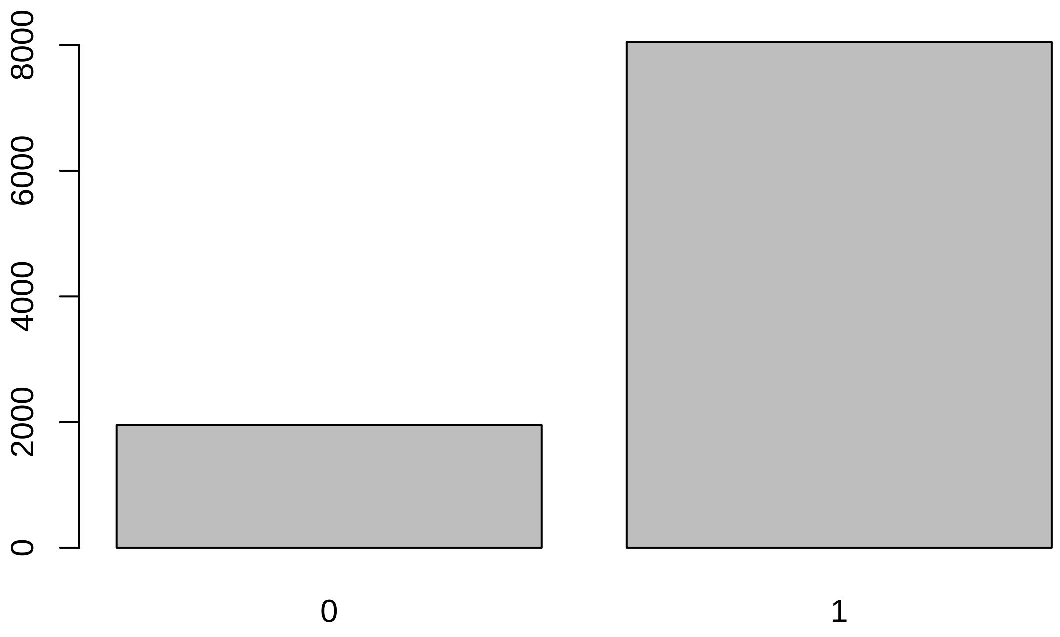
```r
y <- rbernoulli(10000, .8)
y_num <- replicate(10000, 0)
for (i in 1:10000) {
  if (y[i] == TRUE){
    y_num[i] = 1
  }
}
```

(b) For each observation of $Y = y$, simulate from the conditional distribution $X \mid Y = y$.

```r
x <- data.frame(simulated = rbeta(10000, 1 + 4*y_num, 5 - 4*y_num))
```

(c) Plot the samples from the marginal distribution for Y in an appropriate plot of your choosing. What proportion of the samples were equal to 1?

```r
plot(factor(y_num))
```

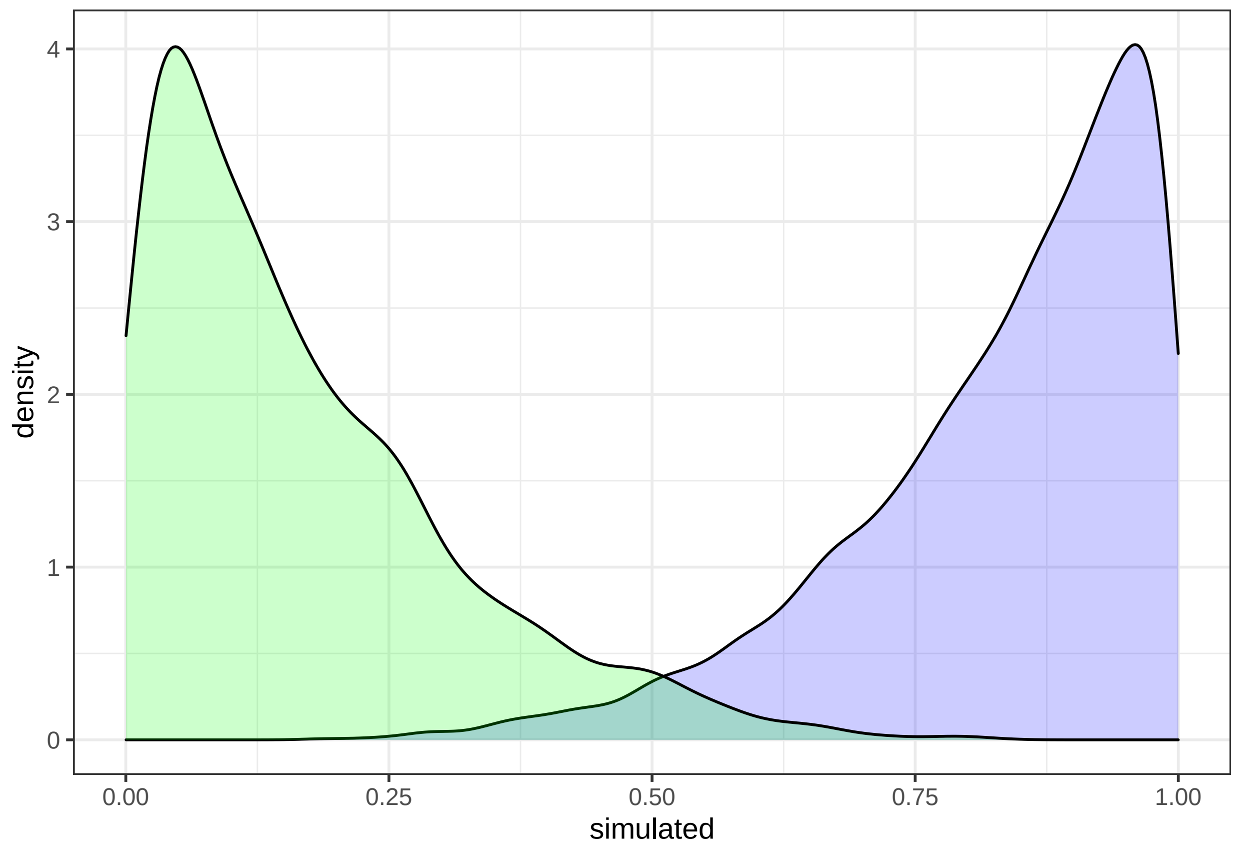


Roughly 80% of the samples are equal to 1.

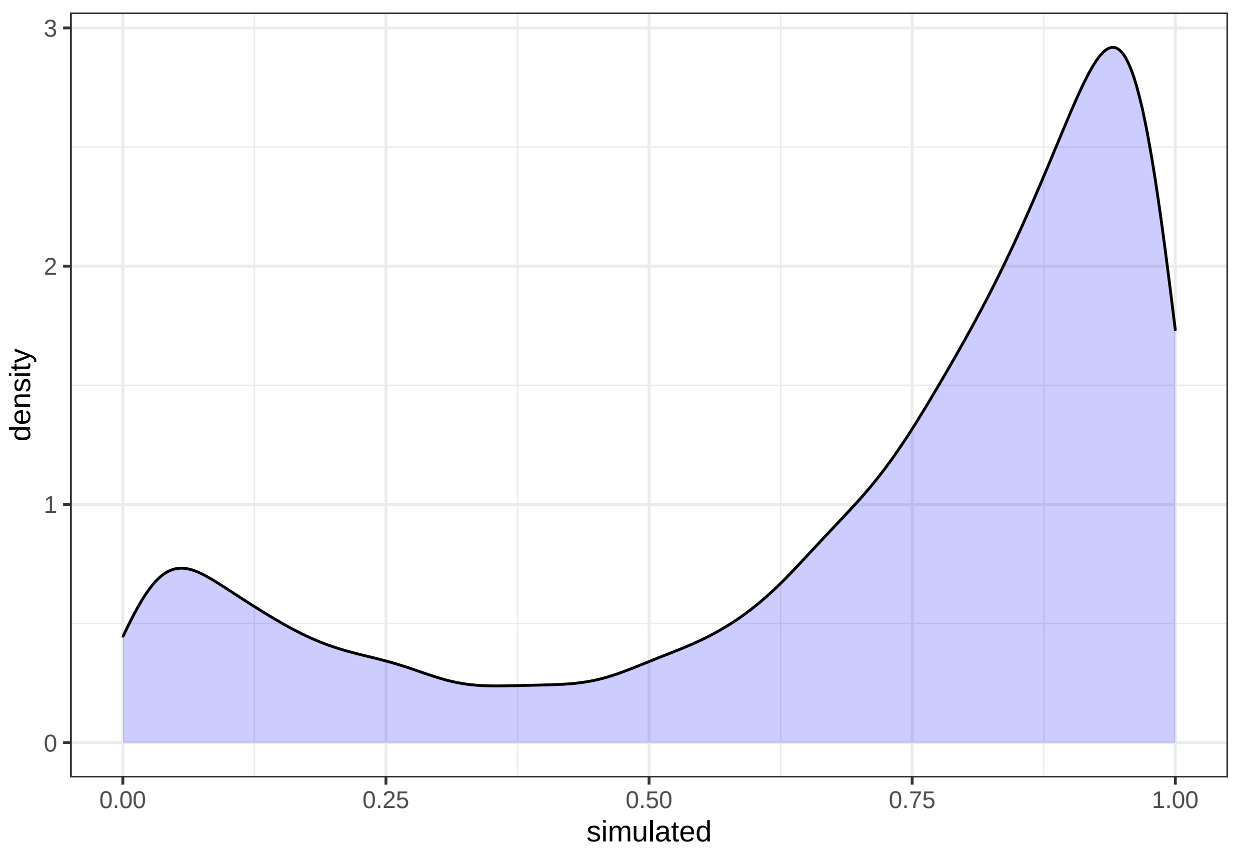
(d) Plot the densities of the conditional distributions $(X \mid Y = 1)$ and $(X \mid Y = 0)$ on the same plot but in different colors. Then on a separate plot, plot marginal distribution for X (ignore which Y was drawn to get that observation for X). Do you recognize what kind of distribution this is?

```r
y_df <- data.frame(y_num, x)
y_1 <- y_df %>%
  filter(y_num == 1)

y_0 <- y_df %>%
  filter(y_num == 0)
```

```r
ggplot() +
  geom_density(data = y_1, aes(x = simulated), alpha = .2, fill = 'blue') +
  geom_density(data = y_0, aes(x = simulated), alpha = .2, fill = 'green')
```



```r
#Plot Marginal Dist for X, appears to be a mixture of 2 beta distributions with weight .2 and weight .8
ggplot(data = x, aes(x = simulated)) +
  geom_density(alpha = .2, fill = 'blue')
```



The takeaway here is that the joint distribution of X and Y may not be recognizable, but we were perfectly able to simulate it from the conditional relationships between X and Y that we knew and the marginal distribution of Y.

Computing Transformations/Functions of the Data

Let's now take a look at a completely different example altogether. Recall from calculus (or check on Wolfram Alpha) that

$$\int_0^1 x^2\,dx = \frac{1}{3}.$$

Suppose you, for some strange reason, forgot the power rule for integration, but you had access to `R` and needed to compute this integral. How could you do this? Well, we know intuitively that integrals are like sums where we multiply the height of the function by a small sliver of the x-axis and then add them up. Let's divide up the interval $[0,1]$ into small, equal-sized section $[a_i, b_i]$ each of size $\Delta x = b_i - a_i$. Pick a point $x_i \in [a_i, b_i]$ in each interval. Then

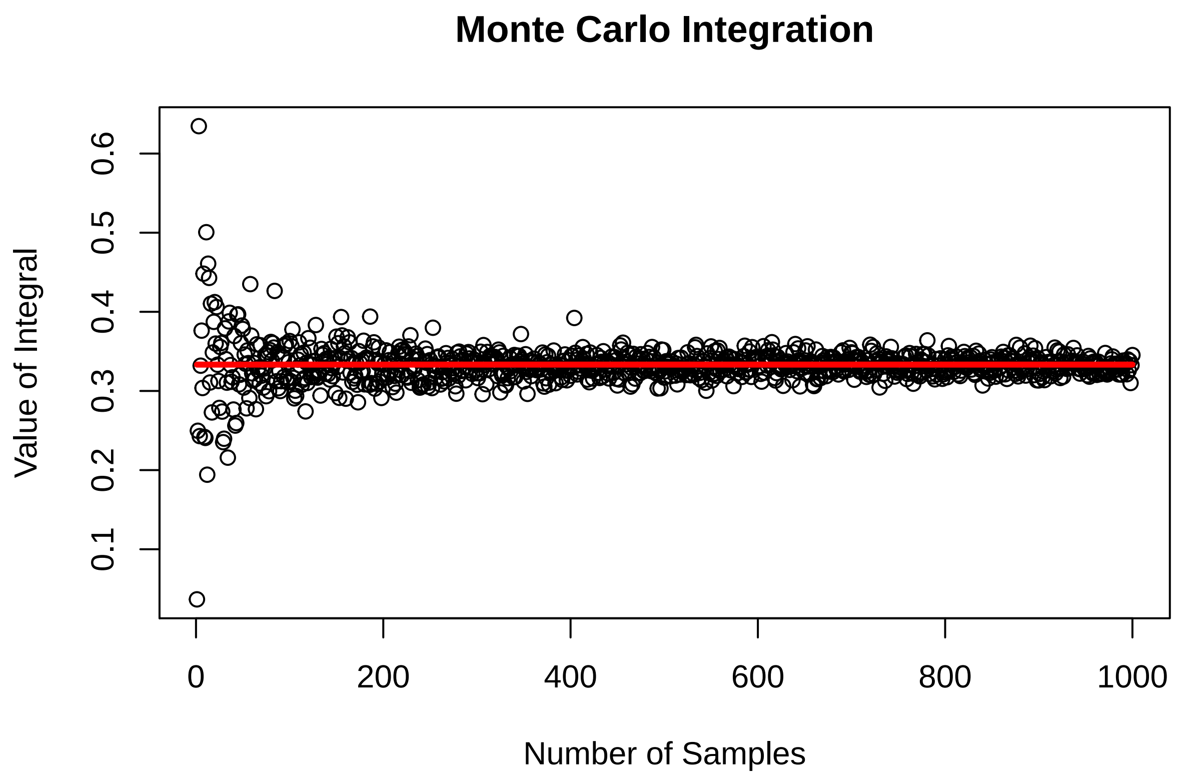
$$\int_0^1 x^2\,dx \approx \sum_{i=1}^{n} x_i^2 \Delta x.$$

Using this as inspiration, suppose $X_1, \ldots, X_n \sim Unif[0,1]$, and assume that each X_i comes from an interval of length $1/n$. Then the above approximation becomes

$$\int_0^1 x^2\,dx \approx \sum_{i=1}^{n} X_i^2 \left(\frac{1}{n}\right) = \frac{1}{n}\sum_{i=1}^{n} X_i^2.$$

So to summarize: to evaluate this integral without doing any calculus, we can just do a Monte Carlo simulation to sample points uniformly on $[0,1]$, square them, and take the average. Let's see how this works in `R`.

```
n <- 1000
approx.vec <- as.numeric(n)
for(i in 1:n){
  approx.vec[i] <- mean(runif(i)^2)
}
plot(1:n, approx.vec, main = "Monte Carlo Integration", ylab = "Value of Integral", xlab = "Number of Sa
lines(x = 1:n, y = rep(1/3, n), col = 'red', lwd=3)
```



Exercise 4 (Not for Completion)

Consider integrating the function $f(x) = (\pi x^5)\sin(x^3\tan(x))$ from -10 to 10. Using Monte Carlo integration to approximate and create a plot like the one above to see how fast it converges to the true value. How many steps do you feel that you needed to get an accurate answer? Compare your answer to Wolfram Alpha. Why do you think the first few steps are so extreme?

Exercise 5 (Not for Completion, Optional Challenge)

This is a fun application of what we've just done. Suppose $X, Y \sim Unif(0, 1)$ are independent of each other.

(a) Draw 100 observations from the joint distribution of (X, Y) (hint: they are independent, so what does this mean for how we can draw each coordinate?). Make sure to plot save your variables `X` and `Y`.

(b) Plot the pairs (X, Y) in the first quadrant of the plane along with the portion of the unit circle that falls in the first quadrant.

(c) Let $D_i = \sqrt{X_i^2 + Y_i^2}$, $E = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(D \leq 1)$ (the indicator that takes on a value of 1 if $D \leq 1$, and 0 otherwise), $P_i = 4E$. Compute each of these values. Does P_i look familiar?

(d) Now repeat the above steps (no need for more plots) for various sizes of n like the examples of Monte Carlo integration. Show in a plot to what value this procedure converges. Do you recognize it?

Visualization and Diagnostics

We have seen a few ways to visualize Monte Carlo simulations, including:

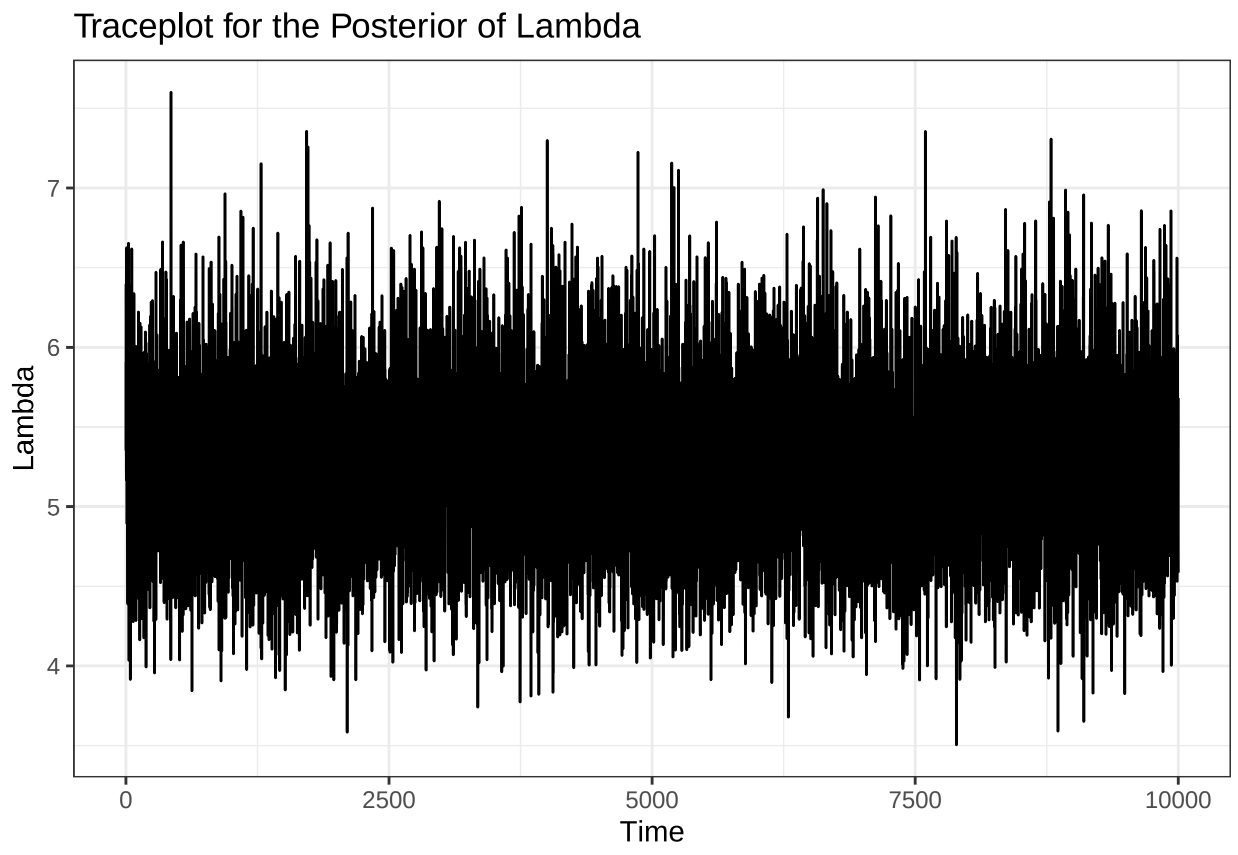
- Histograms
- Density plots
- Convergence plots

The first two are useful ways of describe the distribution from which we sample, while the last one is a great way of seeing if a value we are interested in is converging to anything, and if it is, to what. However, these plots don't quite answer how "well" we are converging to the distribution/quantity of interest. Let's examine a few plots that can help us here.

Exercise 6 (Not for Completion)

Let's revisit the Exponential-Gamma result from Exercises 1 and 2. As we simulate new values, we can plot out the values we simulate "over time" in a **trace plot**. That is, on the x-axis, we plot each simulation, and the y-axis contains the simulated value, in this case samples from the posterior distribution. Plot this for the Exponential-Gamma example below.

```
ggplot(data = lambda_samp, aes(x = 1:length(simulated), y = simulated)) + geom_line() + labs(x = "Time"
```



From this trace plot, consider the following two questions:

- Did the simulation "converge"? Namely, do the values look like they are being simulated from the distribution of interest? Compare this plot to the density plot in Exercise 1.
- We say that a simulation mixes well when it's sampling from the distribution in fairly consistent way and not meandering around the sample space. Does this simulation "mix well"?

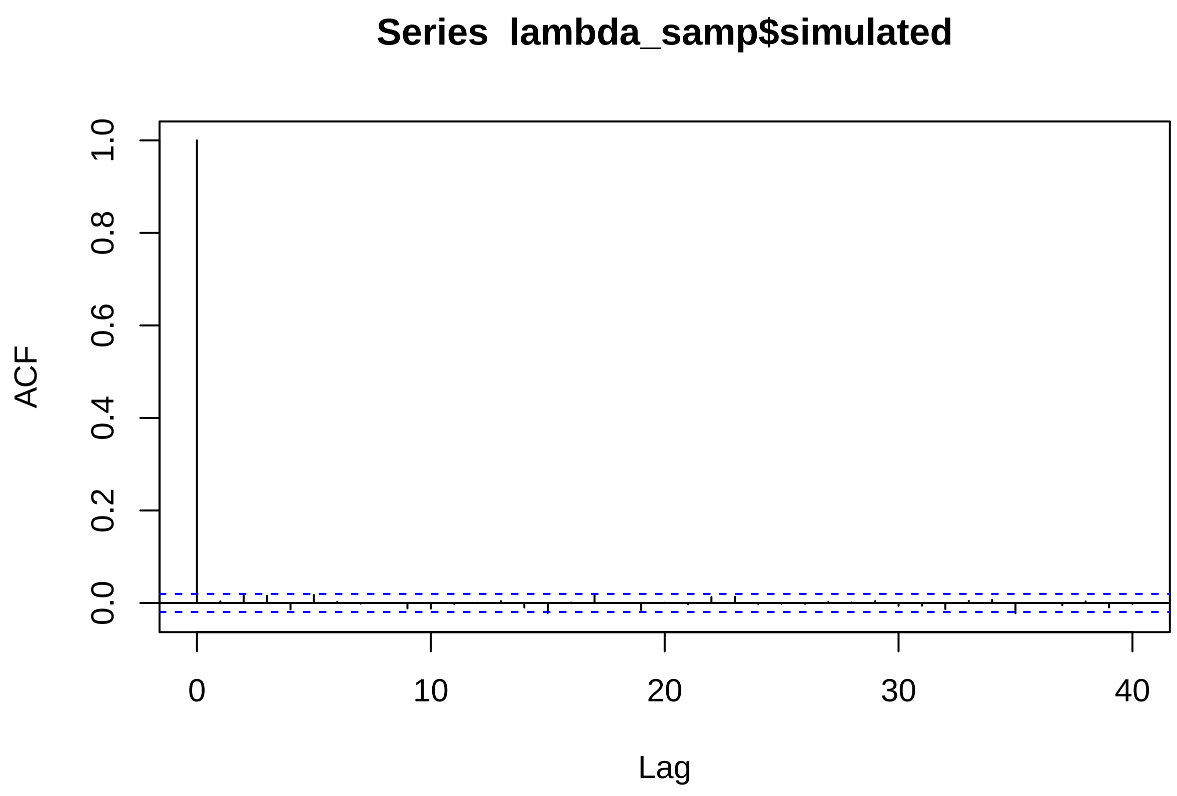
Another relevant notion is of dependence. We may wonder to what extent does one sample depend on previous samples. While it may seem strange, this concept's importance will become more apparent in the next few weeks. To visually assess dependence, we can examine an autocorrelation function (ACF) plot. In a sequence of observations $x_{t}{}_{t=1}^{n}$, the j^{th} autocorrelation is the correlation between an observation and the one that occurred j steps away[1]:

$$\rho_j = \frac{Cov(x_t, x_{t-j})}{\sqrt{Var(x_t)Var(x_{t-j})}}$$

We can draw a plot of this as follows using the `acf()` function in `R`:

```r
acf(lambda_samp$simulated)
```

[1]There is a subtle point we assume here that the autocorrelations only depend on the window between observations. If you're interested, this is called **weak stationarity**, but it's outside the scope of this class.



The height of the bars represent the autocorrelations ρ_j for $j = 1, 2, \ldots, 40$. If the bars are high, then there is significant autocorrelation, indicating dependence in simulated values. Some more questions to consider:

- Why is the first autocorrelation 1?
- Do you think there is dependence in the simulated values based on the ACF plot?

One other notion that is useful is the **effective sample size (ESS)**. The idea is that ideally when we simulate, say 10,000 draws, we want that to represent 10,000 independent draws. When we have dependence, the ESS represents the effective number of independent draws we have out of our total number of draws. This can be easily computed using the `effectiveSize()` function in the `coda` package.

```
#install.packages("coda")
library(coda)
```

```
## Warning: package 'coda' was built under R version 4.0.4
```

```
##
## Attaching package: 'coda'
```

```
## The following object is masked from 'package:rstan':
##
##     traceplot
```

```r
effectiveSize(lambda_samp$simulated)
```

```
##     var1
## 9234.999
```

Why do you think we get this ESS for the Exponential-Gamma example?

Simulating from Distributions Indirectly

In this last example, we will combine many of the elements of previous discussions together. The following example involves simulating two random variables that *depend on each other*:

$$x_t \mid y_{t-1} \sim N(\rho y_{t-1}, 1-\rho^2) \qquad y_t \mid x_t \sim N(\rho x_t, 1-\rho^2)$$

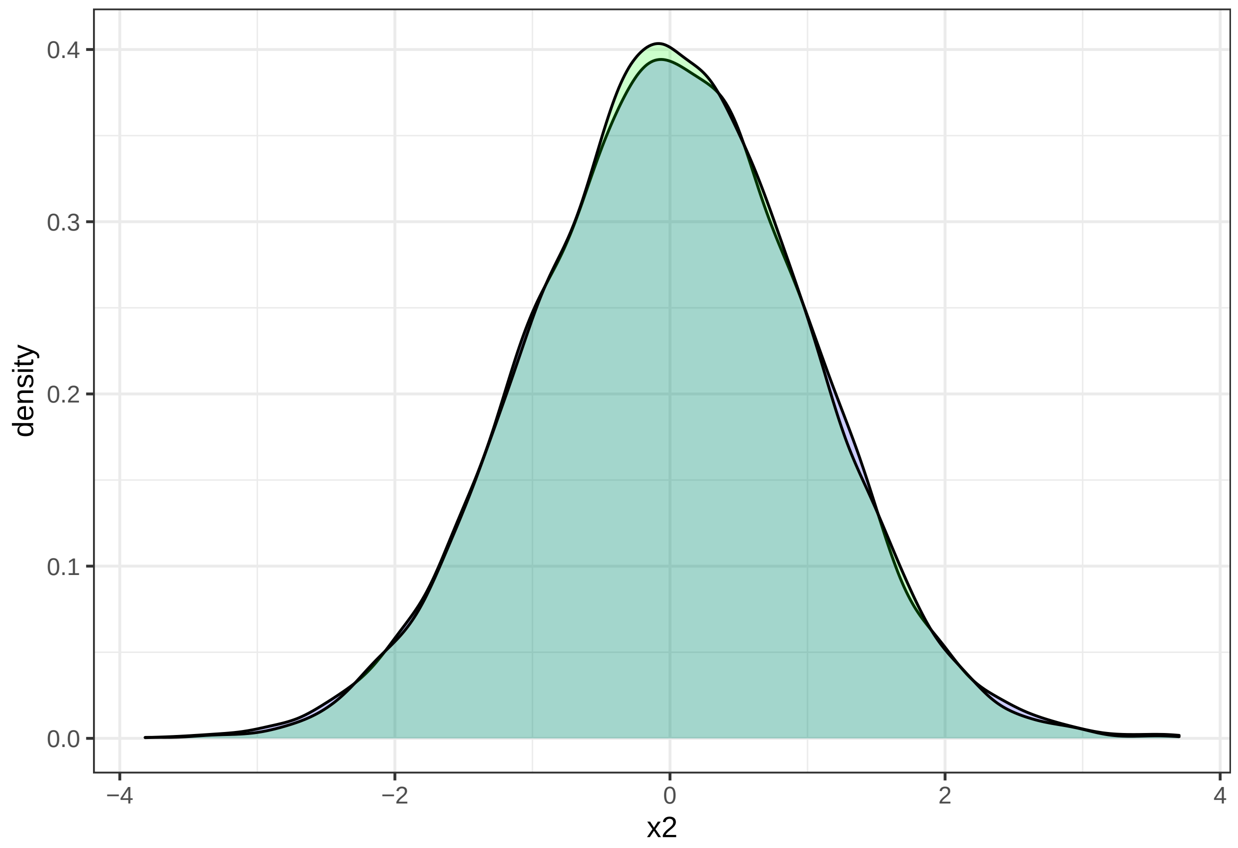
While this may look strange, it turns out this gives us the ability to sample indirectly from the joint distribution for (X, Y) even though we have no idea what it is. The function below is all we will need to run the simulation. You'll create plots below to analyze the output.

```r
mcEx <- function(n, rho, start, plt = FALSE) {
  # Function that prouces simulations for the above model.
  # The output is a list with each entry including the output.
  # n: number of simulations
  # rho: mysterious parameter
  # start: initial point of the simulation
  # plt: if TRUE, plot is output to show

  x1 <- as.numeric(n)
  x2 <- as.numeric(n)
  x1[1] <- start
  x2[1] <- rnorm(n=1, rho*x1[1], sqrt((1-rho^2)))

  for (t in 2:n) {
    x1[t] <- rnorm(n=1, mean = rho*x2[t-1], sd = sqrt((1-rho^2)))
    x2[t] <- rnorm(n=1, mean = rho*x1[t], sd = sqrt((1-rho^2)))
  }

  if (plt == TRUE) {
    x1.plt <- rep(x1, each = 2)[-(2*n)]
    x2.plt <- rep(x2, each = 2)[-1]

    plot(x1.plt, x2.plt)
    lines(x1.plt, x2.plt)
  }

  return(list(x1 = x1, x2 = x2))
}
```

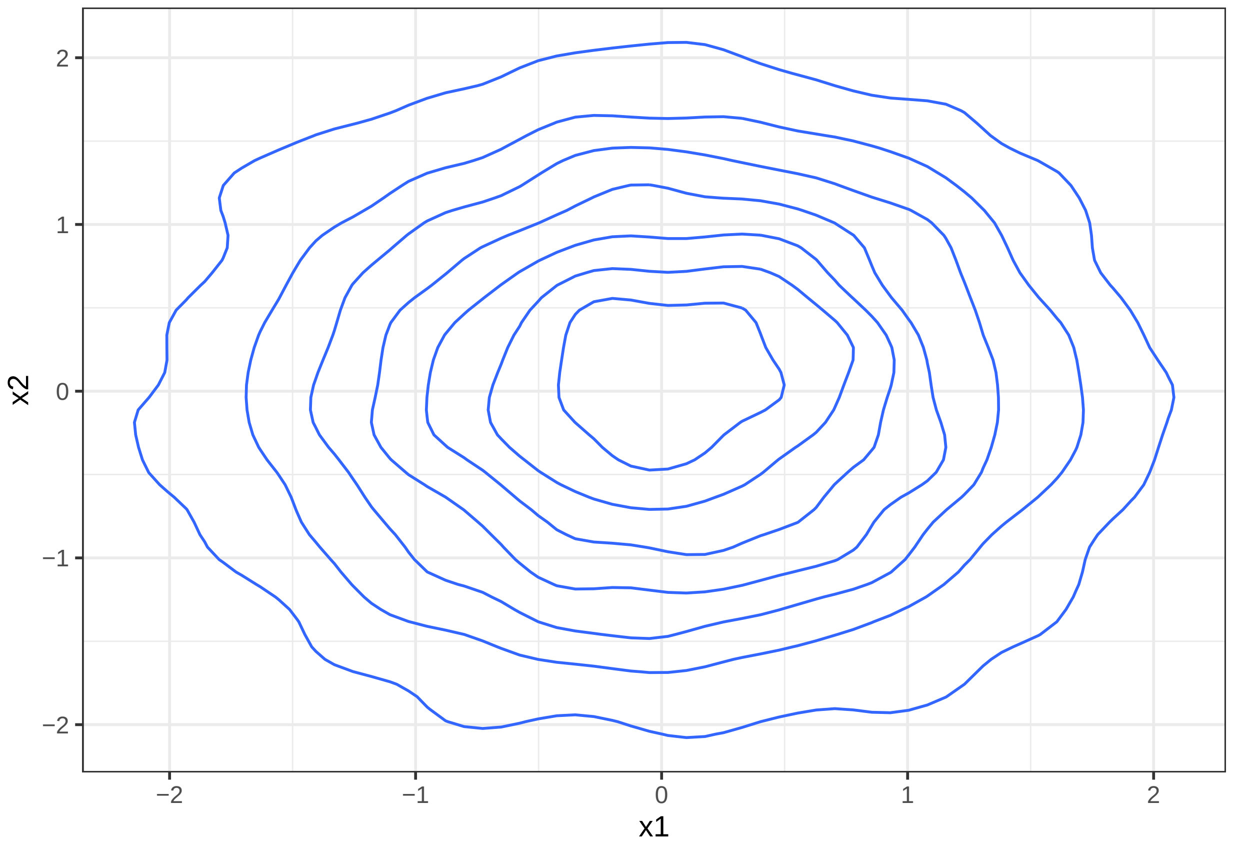
Exercise 7

(a) Run the function `mcEx` with parameters `n = 10000, rho = 0, start = 0, plt = FALSE`. Make density plots of the marginal, and create a 2d density using the `geom_density_2d()` (or an equivalent function). Can you recognize the distribution?

```r
data <- data.frame(mcEx(n = 10000, rho = 0, start = 0, plt = FALSE))

ggplot(data = data) +
  geom_density(aes(x = x2), alpha = .2, fill = "blue") +
  geom_density(aes(x = x1), alpha = .2, fill = "green")
```



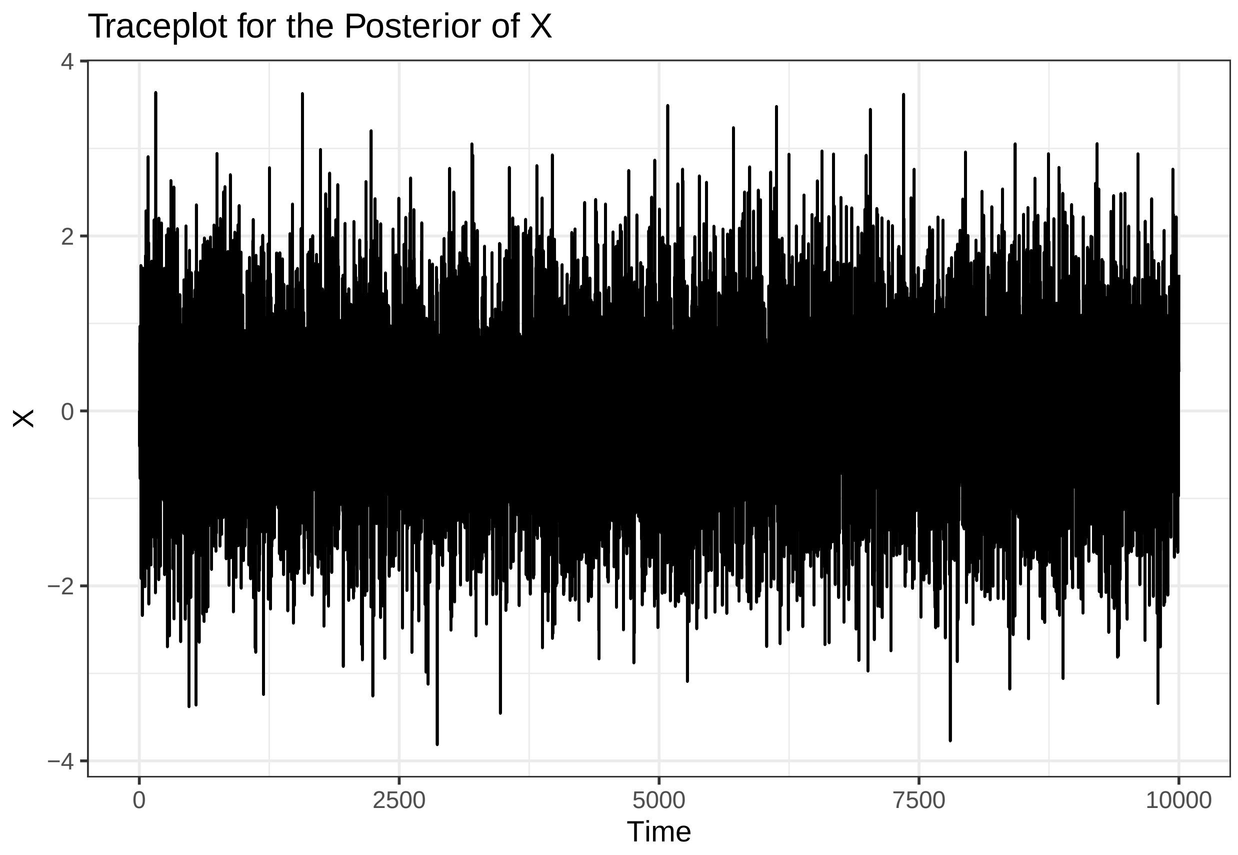
```r
ggplot(data = data) +
  geom_density2d(aes(x = x1, y = x2))
```



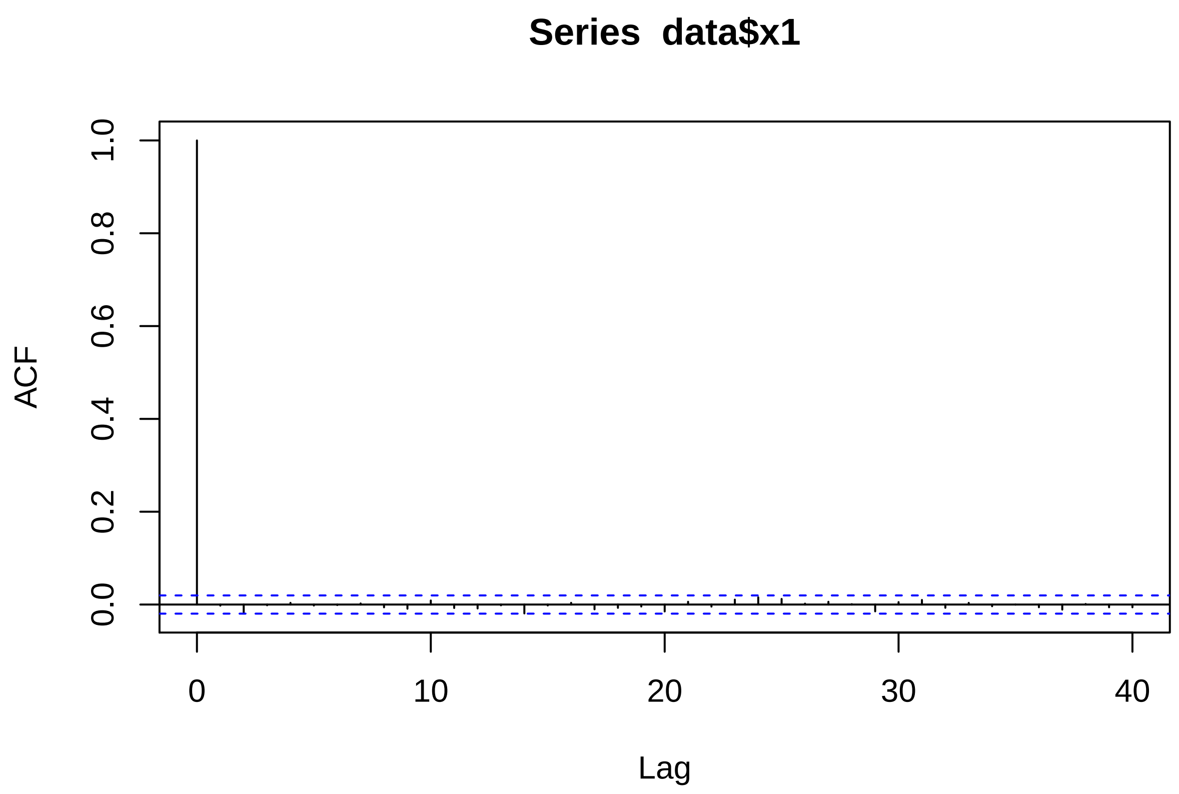
Seems to be a bell-shaped curve marginally, and a symmetric distribution around (0,0) in the bivariate plot. Based on this, we can't conclude much more, but it's possible to prove that this is a bivariate normal distribution with standard normal marginals that have correlation ρ.

(b) Make trace plots and ACF plots for both X and Y. Comment on the plots, noting whether it converges, whether you think the simulations mix well, and the dependence on random draws.

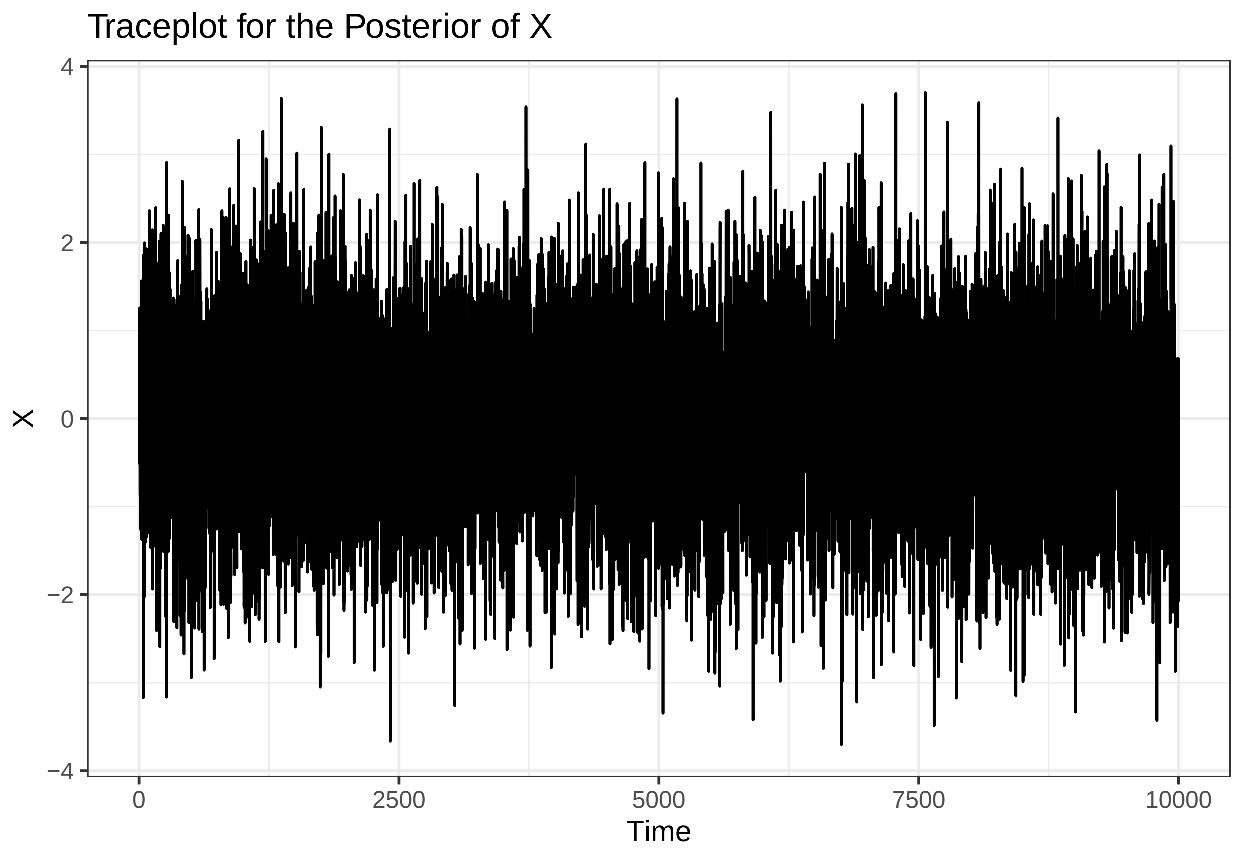
```
ggplot(data = data, aes(x = 1:length(x1), y = x1)) + geom_line() + labs(x = "Time", y = "X", title = "T
```


Traceplot for the Posterior of X
X
Time
4
2
0
−2
−4
0
2500
5000
7500
10000

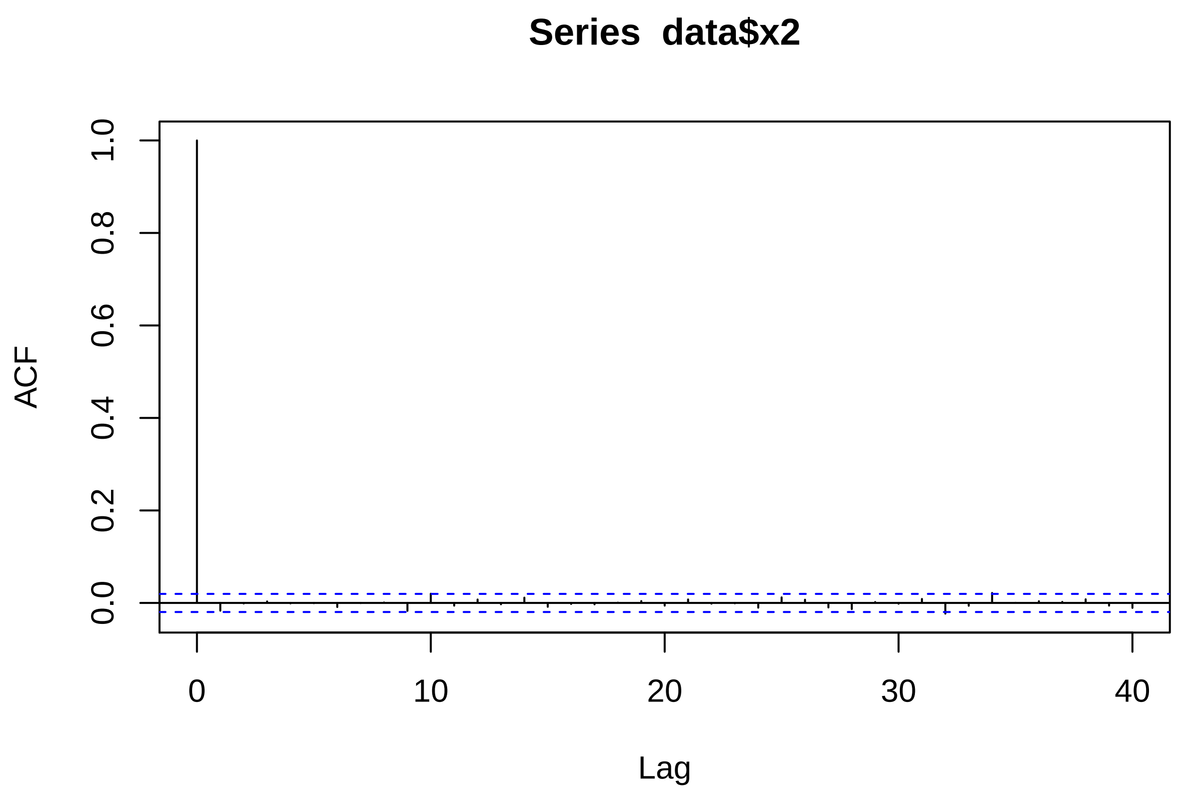
```
acf(data$x1)
```



```
ggplot(data = data, aes(x = 1:length(x2), y = x2)) + geom_line() + labs(x = "Time", y = "X", title = "Ti
```


Traceplot for the Posterior of X
4
2
0
−2
−4
X
0
2500
5000
7500
10000
Time

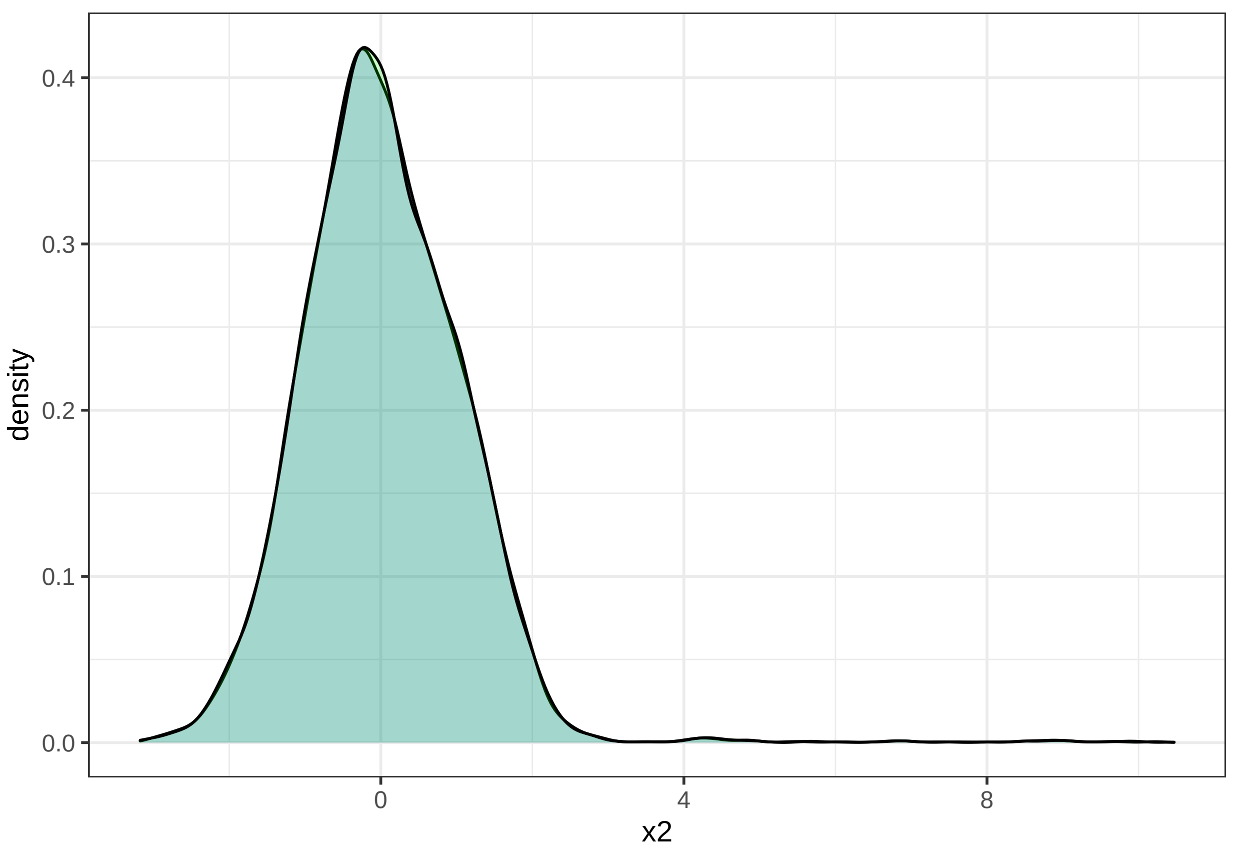
```r
acf(data$x2)
```



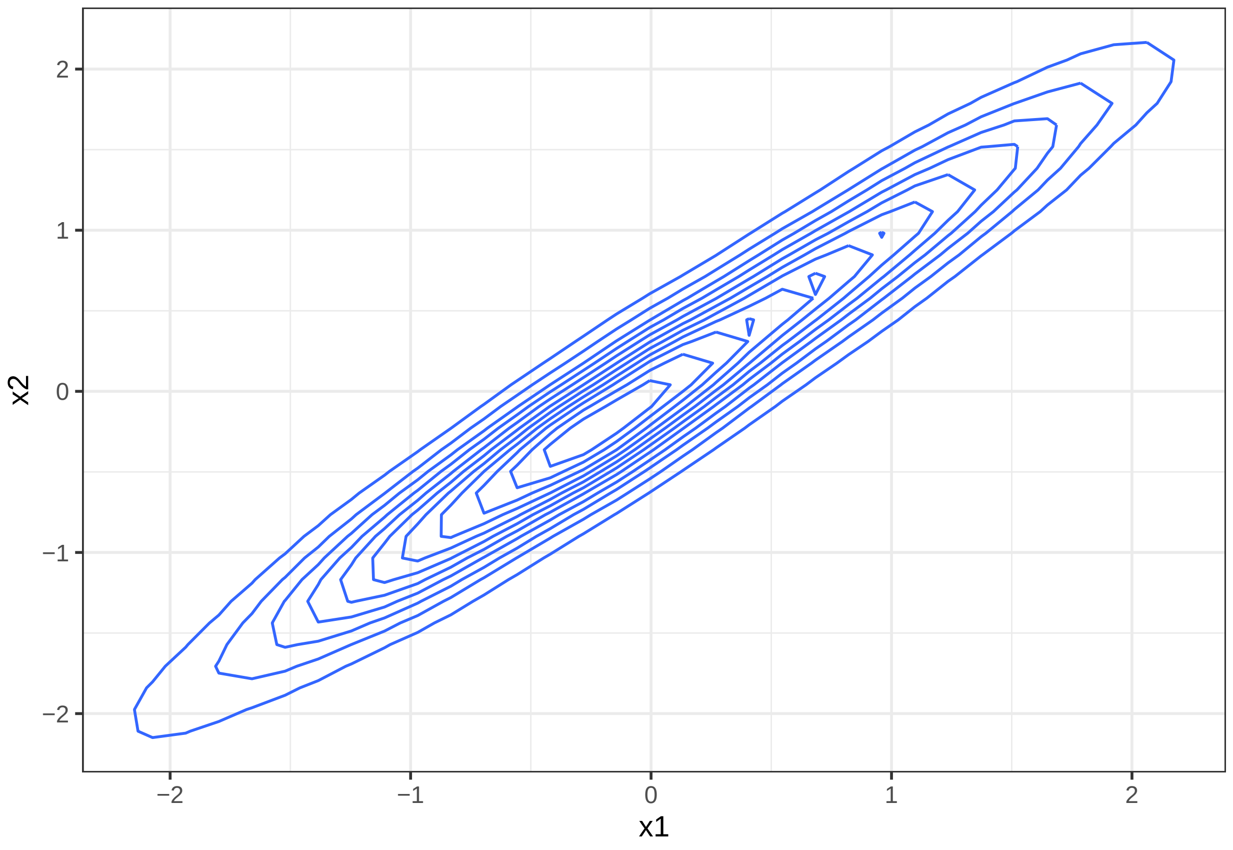
It appears to converged immediately and mixes well. The ACF plot shows almost no dependence.

(c) Repeat exercise (a) with parameters `n = 10000, rho = 0.99, start = 10, plt = FALSE`.

```r
datac <- data.frame(mcEx(n = 10000, rho = .99, start = 10, plt = FALSE))

ggplot(data = datac) +
  geom_density(aes(x = x2), alpha = .2, fill = "blue") +
  geom_density(aes(x = x1), alpha = .2, fill = "green")
```



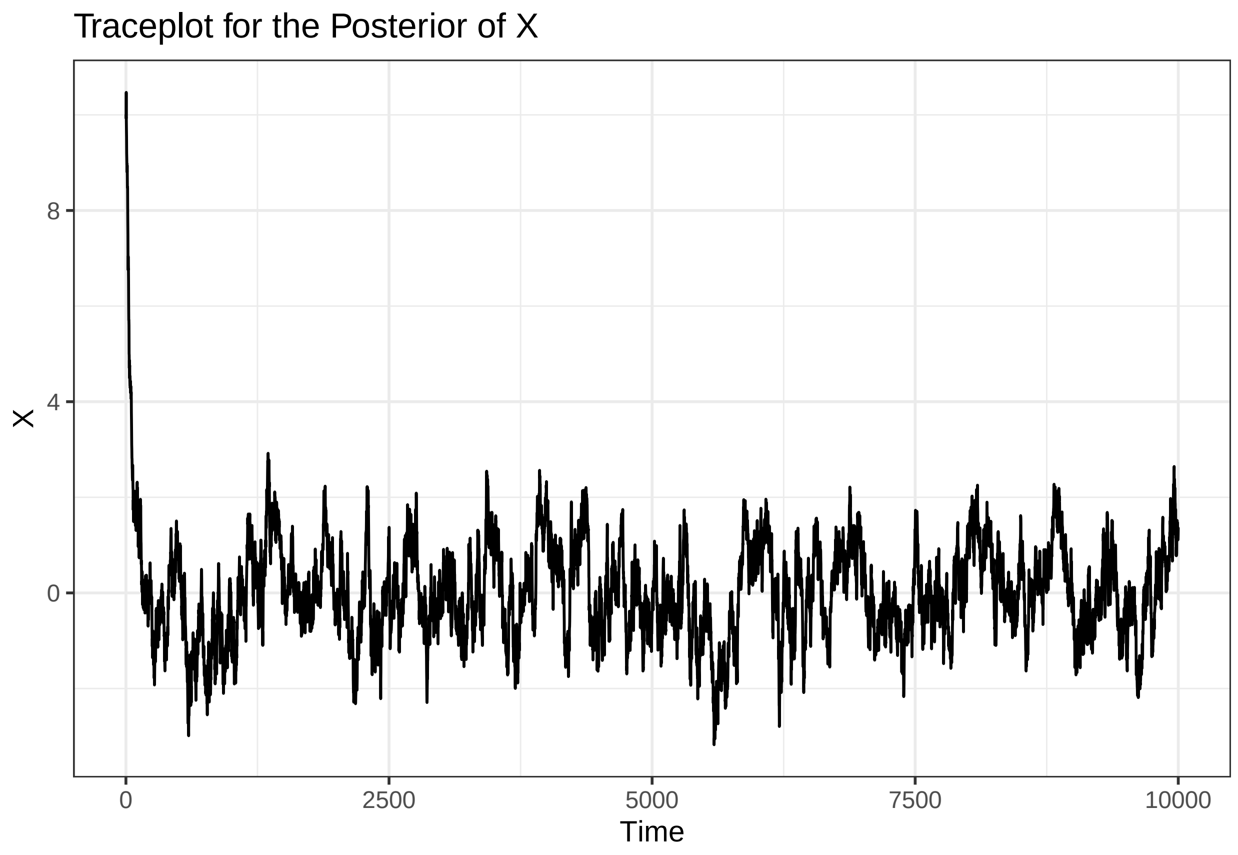
```r
ggplot(data = datac) +
  geom_density2d(aes(x = x1, y = x2))
```



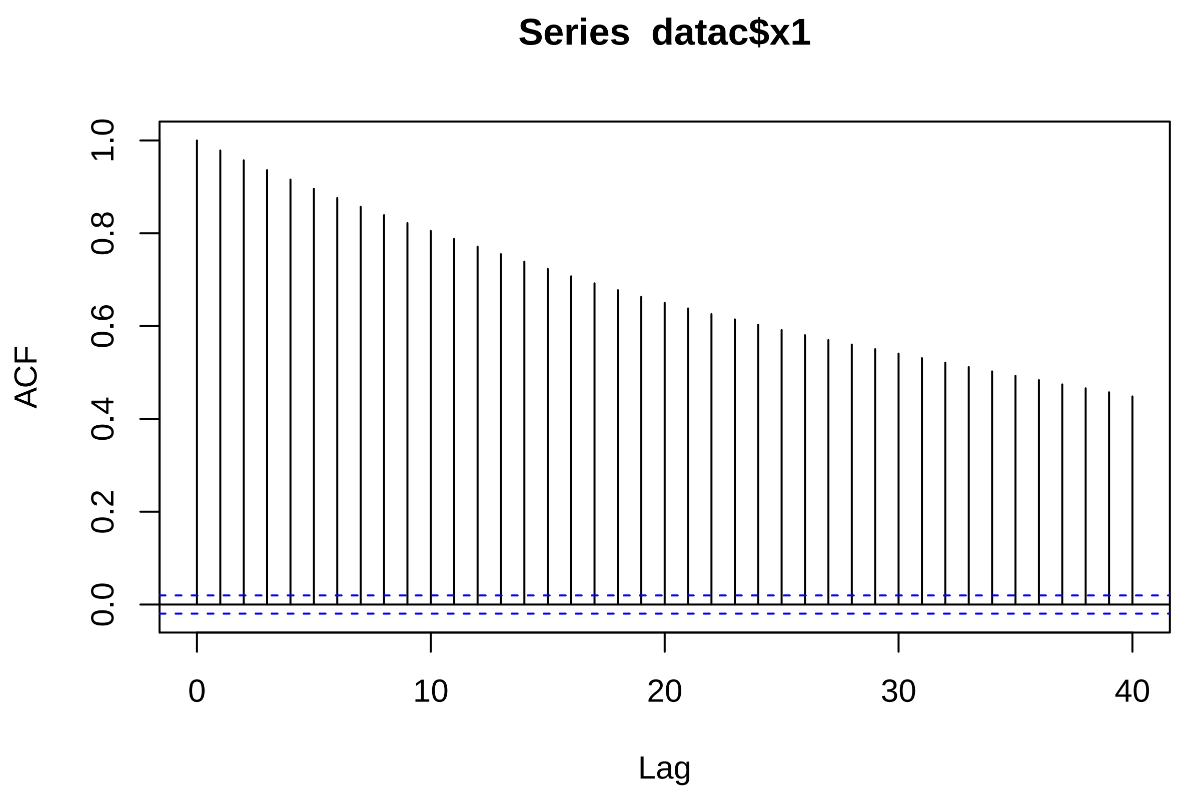
It’s still a bivariate normal distribution, but it is highly correlate as we can see from the bivariate plot.

(d) Repeat exercise (b) using the output of (c).

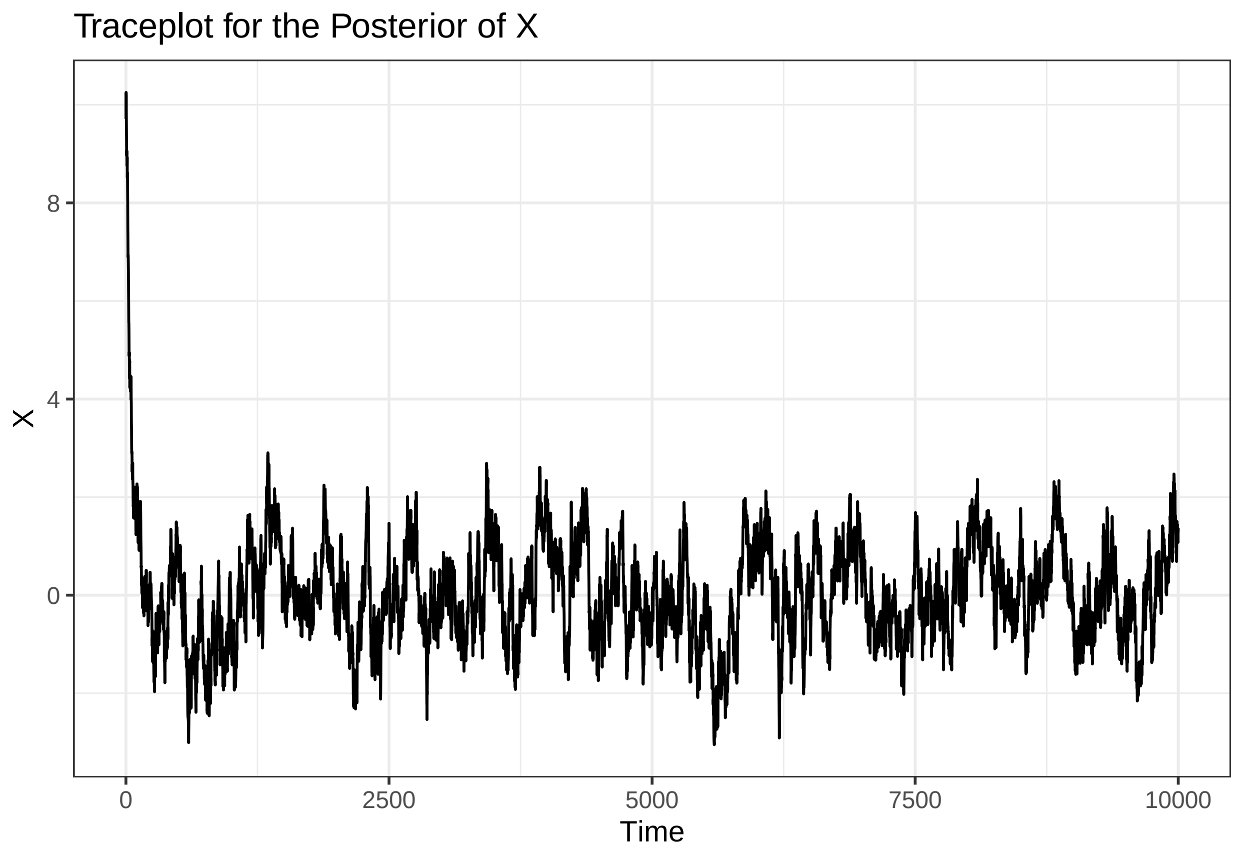
```
ggplot(data = datac, aes(x = 1:length(x1), y = x1)) + geom_line() + labs(x = "Time", y = "X", title = "
```


Traceplot for the Posterior of X
X
8
4
0
0
2500
5000
7500
10000
Time

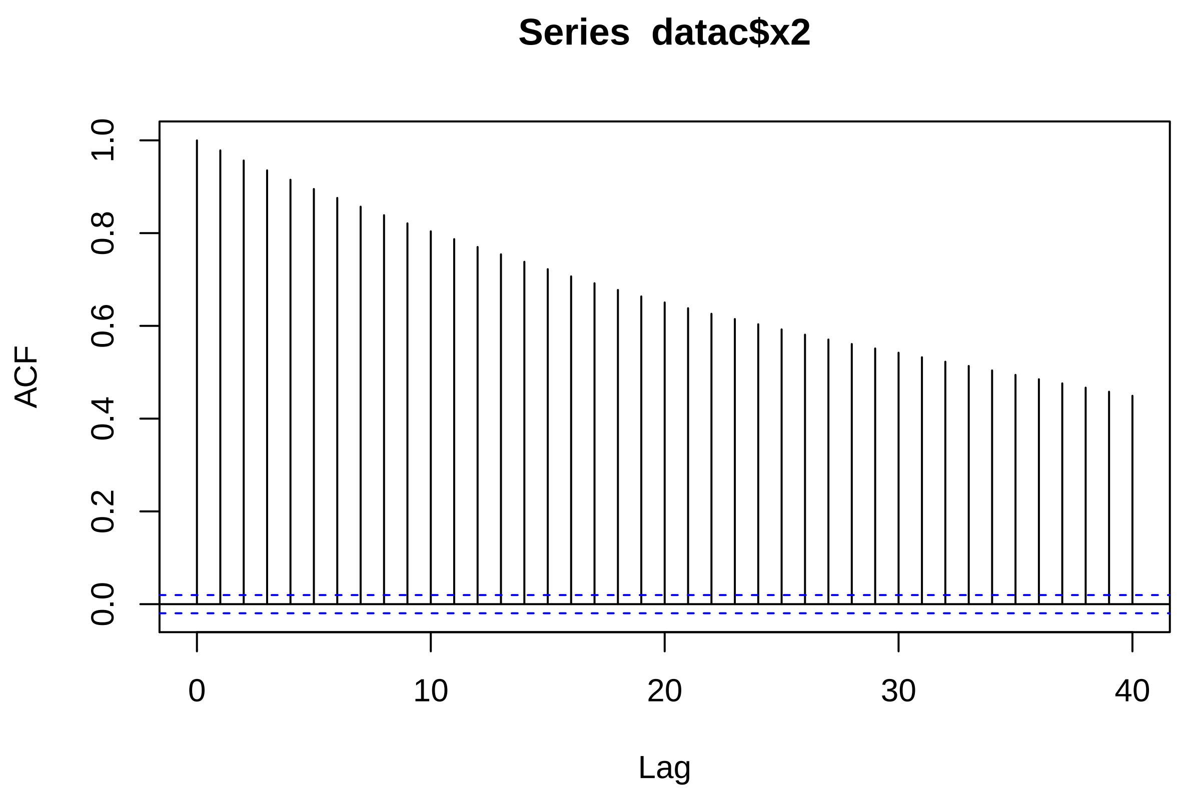
```
acf(datac$x1)
```



```
ggplot(data = datac, aes(x = 1:length(x2), y = x2)) + geom_line() + labs(x = "Time", y = "X", title = "
```


Traceplot for the Posterior of X
X
8
4
0
0
2500
5000
7500
10000
Time

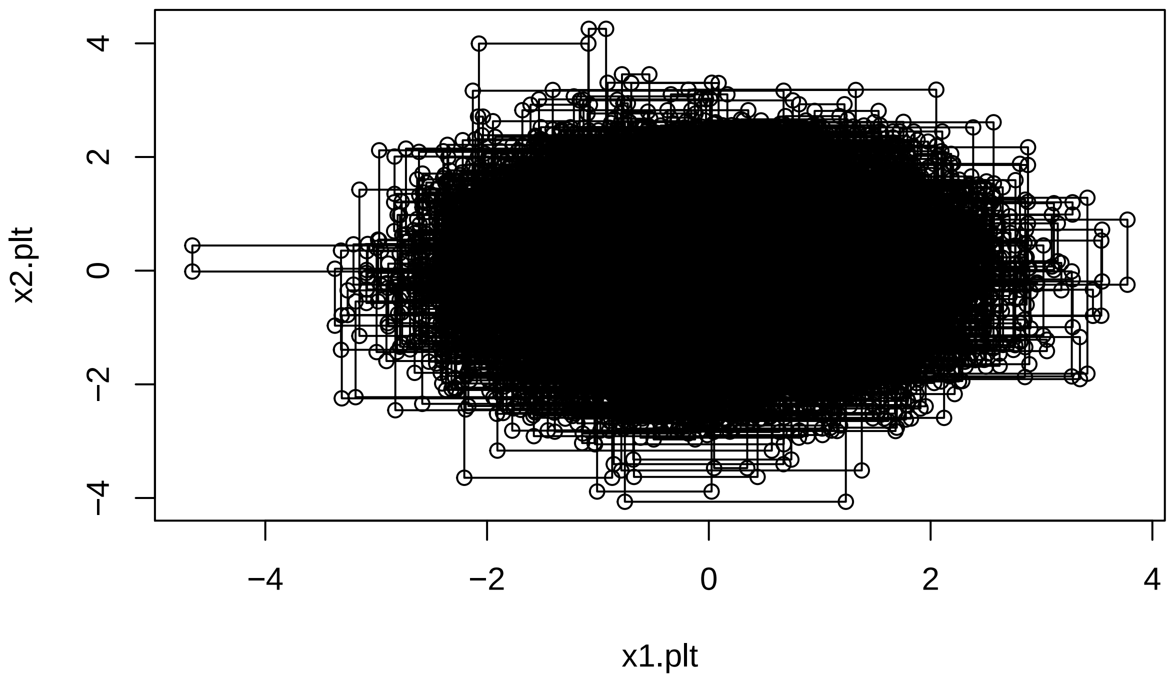
```
acf(datac$x2)
```



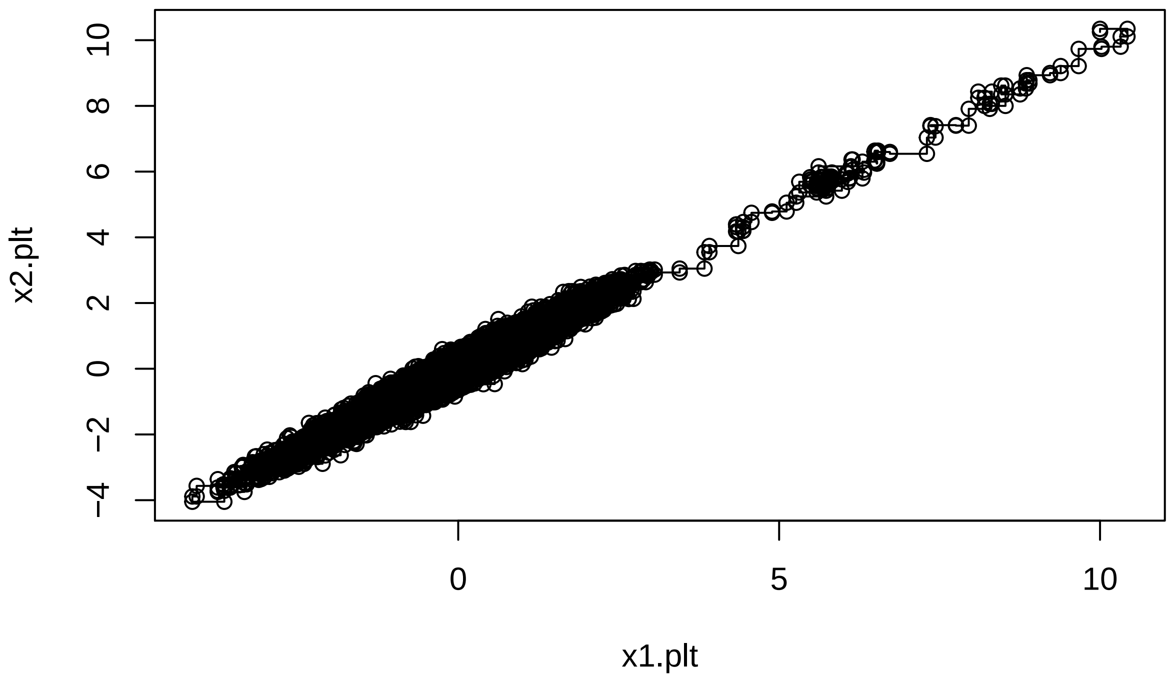
The distribution converges eventually after a what looks like a few hundred step, but the mixing is not great as the simulation wonders around. The ACF reinforces this notion by indicating a high level of dependence between simulations.

(e) (BONUS) Now, run the function `mcEx` using the parameters from parts (a) and (c) but with `plt = TRUE`. Describe the differences you observe in the plots. Can you see the connections between these plots and your answers to the previous parts?

```
data <- data.frame(mcEx(n = 10000, rho = 0, start = 0, plt = TRUE))
```



```
datac <- data.frame(mcEx(n = 10000, rho = .99, start = 10, plt = TRUE))
```



The plots show each update in the X and Y steps. Observe that the steps are always horizontal/vertial (that is, along the axes). Unlike the first plot, which starts at the center and stays there, the second plot starts far away. Due to the high correlation, each update in X and Y is small, and it takes a while to get to the part of the distribution with high concentration.